WebSphere software

IBM

WebSphere Application Server V6 Planning and Design WebSphere Handbook Series

Discusses end-to-end planning for WebSphere implementations

Provides best practices

Includes a complex topology walk-through

Hernan Cunico
Leandro Petit
Michael Asbridge
Derek Botti
Venkata Gadepalli
William Patrey
Noelle Jakusz

Redbooks

ibm.com/redbooks

International Technical Support Organization

WebSphere Application Server V6 Planning and Design, WebSphere Handbook Series

March 2005

SG24-6446-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xix.

First Edition (March 2005)

This edition applies WebSphere Application Server V6.0.

Contents

Notices

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- eserver®
- eserver®
- Redbooks (logo) ™
- alphaWorks®
- e-business on demand™
- eServer™
- ibm.com®
- iSeries™
- i5/OS™
- pSeries®
- z/OS®
- zSeries®
- AIX®
- Balance®
- ClearCase MultiSite®
- ClearCase®
- ClearQuest®
- Cloudscape™
- CICS®
- DataJoiner®
- Domino®
- DB2 Connect™
- DB2®
- Informix®
- IBM®
- IMS™
- Lotus®
- OS/390®
- OS/400®
- Power PC®
- PureCoverage®
- Purify®
- Quantify®
- Rational Rose®
- Rational Suite®
- Rational Unified Process®
- Rational®
- Redbooks™
- Redbooks (logo)™
- RequisitePro®
- RACF®
- RS/6000®
- RUP®
- S/390®
- SecureWay®
- SoDA®
- Tivoli®
- WebSphere®
- XDE™

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Intel Inside (logos are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook discusses the planning and design of industrial strength WebSphere Application Server V6 environments. The content of this redbook is oriented to IT architects and consultants who require assistance when planning and designing from small to large and complex implementations.

This redbook addresses the new packaging and features incorporated in WebSphere Application Server V6, including technology options, product architecture, and integration with other products. It also covers the most common implementation topologies and addresses in detail the planning for specific tasks and components that conform to the WebSphere Application Server environment.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.

Figure 1 Team who wrote this book. First row from left to right, Venkata Gadepalli, Noelle Jakusz, and Michael Asbridge. Second row from left to right, Hernan Cunico, William Patrey, Derek Botti, and Leandro Petit.

Hernan Cunico is a Consulting IT Specialist Certified and WebSphere® software specialist at the ITSO, Raleigh Center. He writes extensively and teaches IBM classes on WebSphere Commerce and WebSphere Application Server. Hernan has 9 years of experience in the Information Technology, and e-business consulting areas. His areas of expertise also include networking, Internet security, e-business and e-commerce solutions architecture.

Leandro Petit is an Advisory System Software Specialist Professional in IBM Argentina. He has 12 years of experience in iSeries™ customer support and 4 years of experience in WebSphere Application Server and WebSphere Portal Server support areas. He holds a degree in Information Technology from the Universidad Argentina de la Empresa.

Michael Asbridge is an Accredited Advisory IT Specialist for IBM UK. He has 6 years of experience in the Application Development - Application Integration discipline and specialization. His areas of expertise include J2EE Application Development and WebSphere Application Server administration. He is a Sun Certified Java™ 2 Programmer. He holds a Master of Science in Computer Science from the University of Birmingham, UK.

Derek Botti is an Applications/Middleware Team lead for the High Performance On Demand Services team in IBM Global Services US. He has over 10 years of experience in the IT industry, with six years of WebSphere Application Server administration experience. He holds a degree in Computer Science from the University of Arkansas, Little Rock. His areas of expertise include High Volume, High Availability Web sites, Internet commerce, and e-business solutions.

Venkata Gadepalli is a member of the WebSphere Enablement Team based in Raleigh. He has more than 7 years of experience in the IT field and has been involved in customer engagements involving WebSphere Portal Server and WebSphere Application Server for the last three years. He has authored numerous papers and has spoken at many conferences on topics involving WebSphere Portal Server and how it integrates with other backends.

William Patrey is an Advisory IT Specialist in the US. He has 9 years of experience in the Information Technology field. He holds a Master's degree in History from Virginia Tech. His areas of expertise include application development, performance testing, and networking.

Noelle Jakusz is an IT Architect at Probitas Technologies, LLC. in Morristown, TN. She currently teaches the WebSphere Application Server V5.x Administration and Application Development curriculum through Right Source Learning Services (an IBM Training Partner). She also teaches and consults on engagements including SOA with Web services and Process Choreographer. She has 7 years of experience in design and development of WebSphere-based and Enterprise applications, including process modeling, implementation and choreography.

Thanks to the following people for their contributions to this project:

Peter Kovari
ITSO, Raleigh Center

Nicolai Nielsen
IBM Denmark

Klemens Haegele
IBM Germany

Partha Sarathy Momidi
IBM India

Sandhya Kapoor
IBM US

Become a published author

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Part 1

Getting started

This part provides a product overview of the WebSphere Application Server and the development tools. It discusses how these are positioned within the WebSphere product family as well as its architecture and integration with other products. Chapters in this section cover the technology options that are available and how WebSphere Application Server V6 has incorporated these technologies. In addition, it provides details about the new features and functionalities in this relase as well as those that has been deprecated.

1

Introduction to WebSphere Application Server V6

IBM WebSphere is the leading software platform for e-business on demand™. Providing comprehensive e-business leadership, WebSphere is evolving to meet the demands of companies faced with challenging business requirements, such as the need for increasing operational efficiencies, strengthening customer loyalty, and integrating disparate systems. WebSphere provides answers in today's challenging business environments.

With IBM WebSphere you can build business-critical applications for the Web. WebSphere includes a wide range of products that help you develop and serve Web applications. These products are designed to make it easier for customers to build, deploy, and manage dynamic Web sites more productively.

This chapter looks at the new WebSphere Application Server V6 for distributed platforms and includes the following sections:

- Product overview
- WebSphere product family
- WebSphere for distributed platforms
- Packaging
- Supported platforms and software

1.1 Product overview

WebSphere Application Server is a high-performance and scalable transaction engine for dynamic e-business applications. The Open Services Infrastructure allows companies to deploy a core operating environment that works as a reliable foundation that is capable of handling high volume secure transactions and Web services.

WebSphere continues the evolution to a single Web services-enabled, Java 2 Enterprise Edition (J2EE) application server and development environment that addresses the essential elements that are needed for an on demand operating environment.

The potential value of J2EE is tremendous. Some of the benefits of J2EE are:

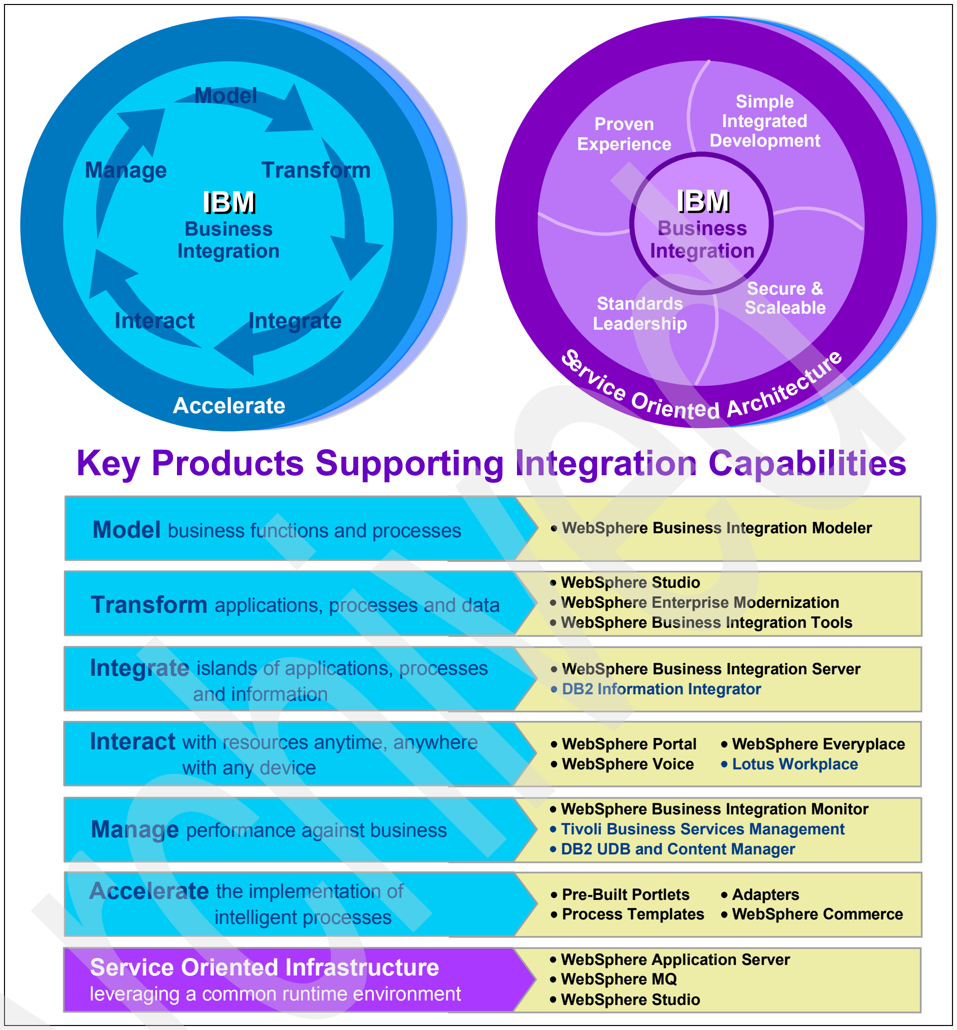
- It includes a simplified architecture that is based on standard components, services, and clients which takes advantage of the write-once, run-anywhere Java technology.
- It contains services that provide integration with existing systems, including Java DataBase Connectivity (JDBC), Java Message Service (JMS), J2EE Connector Architecture (JCA), Java Interface Definition Language (Java IDL), the JavaMail API, and Java Transaction API (JTA and JTS) for reliable business transactions.
- It provides application development that is focused on unique business requirements and rules, rather than common application aspects, such as security and transaction support, improving productivity and shortening development cycles.
- It provides scalability to meet demand by distributing containers across multiple system and by using database connection pooling, for example.
- It gives you a better choice of application development tools and components from vendors providing standard solutions.
- It provides a flexible security model that provides single signon support, integration with legacy security schemes, and a unified approach to securing application components.

1.2 WebSphere product family

The WebSphere platform forms the foundation of a comprehensive business solutions framework. Its extensive offerings are designed to solve the problems of companies of all different sizes. For example, the technologies and tools at the heart of the WebSphere platform you can use to build and deploy the core of an international financial trading application. Yet, it also fits very nicely as the Web site solution for a neighborhood restaurant that simply wants to publish an online menu, hours of operation, and perhaps provide a Web-based table reservation or food delivery system.

WebSphere's complete and versatile nature can sometimes be the source of confusion for people who are trying to make important decisions about platforms and developer toolkits for their business or departmental projects. So, this book can help you get started with understanding the technologies, tools, and offerings of the WebSphere platform.

Figure 1-1 on page 6 shows a high-level overview of the WebSphere platform.



Key Products Supporting Integration Capabilities

Model business functions and processes	• WebSphere Business Integration Modeler
Transform applications, processes and data	• WebSphere Studio • WebSphere Enterprise Modernization • WebSphere Business Integration Tools
Integrate islands of applications, processes and information	• WebSphere Business Integration Server • DB2 Information Integrator
Interact with resources anytime, anywhere with any device	• WebSphere Portal • WebSphere Everyplace • WebSphere Voice • Lotus Workplace
Manage performance against business	• WebSphere Business Integration Monitor • Tivoli Business Services Management • DB2 UDB and Content Manager
Accelerate the implementation of intelligent processes	• Pre-Built Portlets • Adapters • Process Templates • WebSphere Commerce
Service Oriented Infrastructure leveraging a common runtime environment	• WebSphere Application Server • WebSphere MQ • WebSphere Studio

Figure 1-1 WebSphere product family

1.3 WebSphere for distributed platforms

WebSphere Application Server is a suite of servers that implement the J2EE specification. This simply means that you can install and deploy any Web applications that are written to the J2EE specification on any of the servers in the WebSphere Application Server family.

The foundation of the WebSphere brand is the application server. The application server provides the runtime environment and management tools for J2EE and Web Services-based applications. Clients access these applications through standard interfaces and APIs. The applications, in turn, have access to a wide variety of external sources such as legacy systems, databases, and Web services, that can be used to process the client requests.

Figure 1-2 illustrates a product overview of WebSphere Application Server.

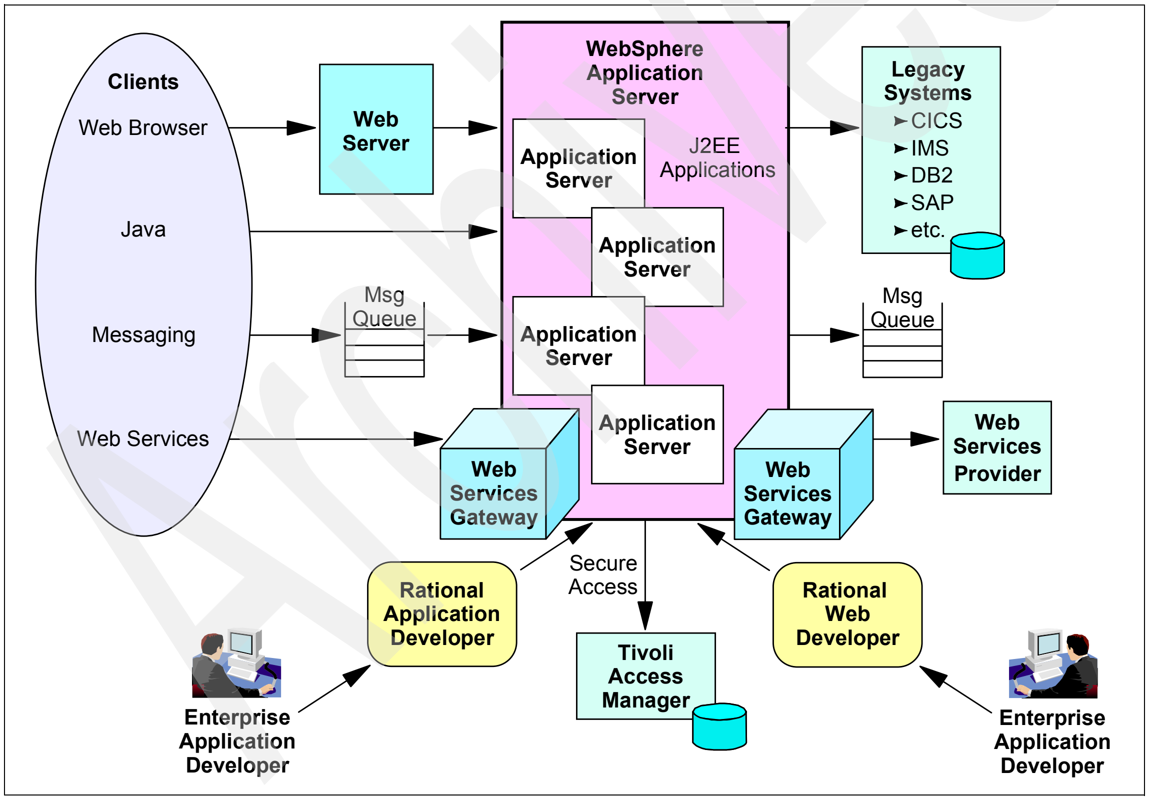


Figure 1-2 WebSphere Application Server product overview

WebSphere Application Server is available in multiple packages to meet specific business needs. It also serves as the base for other WebSphere products, such as WebSphere Commerce, by providing the application server that is required to run these specialized applications.

WebSphere Application Server is available on a wide range of platforms, including UNIX®-based platforms, Microsoft® operating systems, IBM z/OS®, and iSeries. Although branded for iSeries, the WebSphere Application Server products for iSeries are functionally equivalent to those for the UNIX and Microsoft platforms.

1.3.1 Highlights and benefits

WebSphere Application Server provides the environment to run your Web-enabled e-business applications. An application server functions as *Web middleware* or a middle tier in a three-tier e-business environment. The first tier is the HTTP server that handles requests from the browser client. The third tier is the business database (for example, DB2® UDB for iSeries) and the business logic (for example, traditional business applications such as order processing). The middle tier is WebSphere Application Server, which provides a framework for a consistent and architected link between the HTTP requests and the business data and logic.

WebSphere Application Server is intended for organizations that want to take advantage of the productivity, performance advantages, and portability that Java provides for dynamic Web sites. It includes:

- J2EE 1.4 support.
- High-performance connectors to many common back-end systems to reduce the coding effort required to link dynamic Web pages to real line-of-business data.
- Application services for session and state management.
- Web services that enable businesses to connect applications to other business applications, to deliver business functions to a broader set of customers and partners, to interact with marketplaces more efficiently, and to create new business models dynamically.
- The WebSphere Platform Messaging infrastructure to complement and extend WebSphere MQ and application server. It is suitable for those that are currently using the WebSphere Application Server V5 embedded messaging and for those that need to provide messaging capability between WebSphere Application Server and an existing WebSphere MQ backbone.

Platform messaging features include:

- Multiple messaging patterns (APIs) and protocols for message-oriented and service-oriented applications.
- The default messaging provider is a J2EE 1.4 compliant JMS provider.
- The relevant Web services standards that support JAX-RPC APIs.
- Reliable message transport capability.
- Tightly and loosely coupled communications options.
- Intermediary logic (mediations) to adapt message flow intelligently in the network.
- Clustering support to provide scalability and high availability.
- Quality of service options.
- Support for the WebSphere Business Integration programming model which converges functions from workflow, message brokering, collaborations, adaptors, and the application server.
- Full integration within WebSphere Application Server, including security, installation, administrative console, performance monitoring, trace, and problem determination.
- Connectivity with a WebSphere MQ network.

1.4 Packaging

Because different levels of application server capabilities are required at different times as varying e-business application scenarios are pursued, WebSphere Application Server is available in multiple packaging options. Although these packaging options share a common foundation, each option provides unique benefits to meet the needs of applications and the infrastructure that supports them. So, at least one WebSphere Application Server product package will fulfill the requirements of any particular project and the prerequisites of the infrastructure that supports it. As your business grows, the WebSphere Application Server family provides a migration path to higher configurations.

Operating system terminology:

- Distributed: Windows®, UNIX – AIX®, Linux/Intel®, Linux/PPC, zLinux, Solaris, HP-UX
- iSeries: i5/OS and OS/400®
- z/OS: OS/390®

1.4.1 Packaging for distributed platforms

WebSphere Application Server is available in several different packages that are designed to meet a wide range of customer requirements. Each package is designed to provide a tailored environment for a specific set of customers.

WebSphere Application Server - Express

WebSphere Application Server - Express is geared to those who need to get started quickly with e-business. It is specifically targeted at medium-sized businesses or departments of a large corporation. It provides easy use and application development. It contains full J2EE 1.4 support but is limited to a single server environment.

The WebSphere Application Server - Express offering is unique in that it is bundled with an application development tool. Although there are WebSphere Studio and Rational® Developer products that are designed to support each WebSphere Application Server package, these products are normally ordered independently of the server. WebSphere Application Server - Express includes the Rational Web Developer application development tool. It provides a development environment geared toward Web developers and includes support for most J2EE 1.4 features with the exception of EJB and JCA development environments. However, keep in mind that WebSphere Application Server - Express does contain full support for EJB and JCA, so you can deploy applications with them.

WebSphere Application Server V6

The WebSphere Application Server V6 package is the next level of server infrastructure in the WebSphere Application Server family. Although the WebSphere Application Server V6 package is functionally equivalent to the Express package, it differs slightly in packaging and licensing. The development tool included is a trial version of Rational Application Developer, a full J2EE 1.4 compliant development tool.

WebSphere Application Server Network Deployment V6

WebSphere Application Server Network Deployment is a higher level of server infrastructure in the WebSphere Application Server family than WebSphere Application Server. It extends the WebSphere Application Server package to include clustering capabilities, Edge Components, and high availability for distributed configurations. These features become more important at larger enterprises, where applications tend to service a larger customer base, and more elaborate performance and availability requirements are in place.

Application servers in a cluster can reside on the same or multiple machines. A Web server plug-in installed in the Web server can distribute work among clustered application servers. In turn, Web containers running servlets and JSPs can distribute requests for EJBs among EJB containers in a cluster.

The addition of Edge Components provides high performance and high availability features, such as:

- Caching Proxy, which intercepts data requests from a client, retrieves the requested information from the application servers, and delivers that content back to the client. It stores cachable content in a local cache before delivering it to the client. Subsequent requests for the same content are served from the local cache, which is much faster and reduces the network and application server load.
- Load Balancer, which provides horizontal scalability by dispatching HTTP requests among several, identically configured Web server or application server nodes.

Table 1-1 shows the features that are included with each WebSphere Application Server packaging option.

Table 1-1 WebSphere Application Server packaging and license terms

	WebSphere Application Server - Express V6	**WebSphere Application Server V6**	**WebSphere Application Server Network Deployment V6**
Licensing terms	Limited to a maximum of two processors	Unlimited processors	Unlimited processors
WebSphere Application Server	Yes	Yes	Yes
Network Deployment	No	No	Yes
IBM HTTP Server V6 Web server plug-ins	Yes	Yes	Yes
IBM HTTP Server	Yes	Yes	Yes
Application Client (not on zLinux)	Yes	Yes	Yes
Application Server Toolkit	Yes	Yes	Yes
DataDirect Technologies JDBC Drivers for WebSphere Application Server	Yes	Yes	Yes

	WebSphere Application Server - Express V6	**WebSphere Application Server V6**	**WebSphere Application Server Network Deployment V6**
Development tools	Rational Web Developer (single use license)	Rational Application Developer Trial	Rational Application Developer Trial
Database	IBM DB2 Universal Database Express V8.2	IBM DB2 Universal Database Express V8.2 (development use only)	IBM DB2 UDB Enterprise Server Edition V8.2 for WebSphere Application Server Network Deployment
Production ready applications	IBM Business Solutions	No	No
Tivoli® Directory Server for WebSphere Application Server (LDAP server)	No	No	Yes
Tivoli Access Manager Servers for WebSphere Application Server	No	No	Yes
Edge Components	No	No	Yes

1.5 Supported platforms and software

The following tables illustrate the platforms, software, and versions that WebSphere Application Server V6 supports at the time of the writing of this document. For the most up-to-date operating system levels and requirements, refer to the following:

- WebSphere Application Server - Express

  http://www.ibm.com/software/webservers/appserv/express/requirements/

- WebSphere Application Server

  http://www.ibm.com/software/webservers/appserv/was/requirements/

- WebSphere Application Server Network Deployment

  http://www.ibm.com/software/webservers/appserv/was/network/requirements/

1.5.1 Operating systems

Table 1-2 shows the supported operating systems and versions for WebSphere Application Server V6.

Table 1-2 Supported operating systems and versions

Operating Systems	Versions
Windows	Windows 2000 Advanced Server, Server, Professional SP4 Windows Server 2003 Datacenter, Enterprise, Standard Windows XP Professional SP1
AIX	AIX 5.1 Maintenance Level 5100-05 AIX 5.2 Maintenance Level 5200-02, 5200-03
Sun Solaris	Solaris 8 with the latest patch Cluster Solaris 9 with the latest patch Cluster
HP-UX	HP-UX 11iv1 with the latest Quality Pack
Linux® (Intel)	RedHat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
Linux (Power PC®)	RedHat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0

Operating Systems	Versions
zLinux (Supported for WebSphere Application Server Network Deployment V6 only)	RedHat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
i5/OS™ and OS/400	OS/400 5.3, 5.2
z/OS (Supported for WebSphere Application Server Network Deployment V6 only)	z/OS 1.6, 1.5, 1.4 z/OS.e 1.6, 1.5, 1.4

1.5.2 Web servers

Table 1-3 shows the Web servers that WebSphere Application Server V6 supports by platform.

Table 1-3 Supported Web servers by platforms

Web servers	Platforms
IBM HTTP Server V6 (powered by Apache 2.0.x)	Windows, AIX, HP-UX, Solaris, Linux (Intel), Linux (Power PC), zLinux, OS/400, z/OS
IBM HTTP Server V2.0.43 for iSeries (powered by Apache)	OS/400
IBM HTTP Server for z/OS 5.0.1	z/OS
Apache Server 2.0.48	Windows, AIX, HP-UX, Solaris, Linux (Intel), Linux (Power PC), zLinux
Microsoft Internet Information Services 5.0, 6.0	Windows
Lotus® Domino® Enterprise Server (as HTTP server) 6.0.3, 6.5.1	Windows, AIX, HP-UX, Solaris, Linux (Intel), Linux (Power PC), zLinux, OS/400, z/OS
Sun ONE Web Server Enterprise Edition 6.0 SP7	Solaris, Windows
Sun Java System Web Server 6.1 SP1	Solaris, Windows
Covalent Enterprise Ready Server 2.4 Apache 2.0.48 Edition	Windows, HP-UX, Linux (Intel) Solaris, z/OS

1.5.3 Database servers

Table 1-4 shows the operating systems and versions that WebSphere Application Server V6 supports.

Table 1-4 Supported database servers and versions

Databases	Versions
IBM DB2	IBM DB2 Enterprise Server Edition 8.2, 8.1 with FP5 IBM DB2 Workgroup Server Edition 8.2, 8.1 with FP4a, FP5 IBM DB2 Information Integrator 8.2, 8.1 with FP5 IBM DB2 Connect™ 8.2, 8.1 with FP5 IBM DB2 Express 8.2, 8.1 with FP5 IBM DB2 for zSeries® 8, 7 IBM DB2 for iSeries 5.3, 5.3
Cloudscape™	Cloudscape 5.1.6x
Oracle	Oracle Standard/Enterprise Edition 10g Release 1 - 10.1.0.2 Oracle 9i Standard/Enterprise Edition Release 2 - 9.2.0.4 Oracle 8i Standard/Enterprise Edition Release 3 - 8.1.7.4
Sybase	Sybase Adaptive Server Enterprise 12.5.1, 12.0.0.8
Microsoft SQL Server	SQL Server Enterprise 2000 SP 3a
Informix®	Informix Dynamic Server 9.4, 9.3

1.5.4 Directory servers

Table 1-5 shows the directory servers and versions that WebSphere Application Server V6 supports.

Table 1-5 Supported Directory servers and versions

Directory Server	Versions
IBM Directory Server	5.1 5.2
z/OS Security Server	1.6 1.5 1.4
Lotus Domino Enterprise Server (acting as LDAP server)	6.5.1 6.0.3
Sun ONE Directory Server	5.2 5.1 SP3

Directory Server	Versions
Windows Active Directory	2003 2000
NDS eDirectory	8.7.3

2

Technology options

WebSphere Application Server V6 introduces new technologies and adopts the current standards for many existing technologies. This chapter discusses the technologies that are available to applications and architecture which are designed to use WebSphere Application Server V6. It provides you with a high-level discussion of the technologies that are available. This chapter contains the following sections:

- Web services
- Flow languages
- J2EE Connector Architecture
- Java Message Service
- Business Rule Bean Framework
- Information integration

2.1 Web services

Web services describes the APIs that are associated with using World Wide Web technologies to provide application-to-application communication. Web services are self-contained, self-describing modular applications that can be invoked using World Wide Web calls. These applications can be simple requests or far more complex business processes, depending on the needs of the application requestor. This is not necessarily a new trend, but it is one that has become far more formalized of late.

This section presents some discussions on Web services that are provided by WebSphere Application Server V6 and their potential uses within an application environment. For a more detailed overview of Web services, refer to *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891, or the IBM developerWorks Web services tutorial that is available at:

`http://www.ibm.com/developerworks/websphere/zones/webservices/`

2.1.1 Static and dynamic Web services

There are two common implementation methods when developing Web services enabled applications. The *static* Web service is one that has a direct relationship with another application using a fixed URL over HTTP or HTTPS. There is no alternative path. If the endpoint is unavailable, the service is effectively down. When implementing a Web service provider for a static Web service, you should minimize the single point of failure that this service provides.

The other common Web service implementation is the *dynamic* Web service. In a dynamic Web service, a provider is registered with a service broker called the Universal Description Discovery and Integration (UDDI) registry. This registry can be private for an organization (such as for use in intranet applications only) or a public service broker, for use in Enterprise applications that could involve more than one business entity. Web services registered in a UDDI registry use Web services Description Language (WSDL) to describe their Web services. A service provider would thus publish the service to a registry via WSDL, and a service requestor would locate the service using a UDDI lookup. After the service is found, the requestor would bind to the provider using a Simple Object Access Protocol (SOAP) call.

2.1.2 JAX-RPC

JAX-RPC, also known as JSR-101, is the Java API for XML-based Remote Procedure Calls (RPC.) This API allows Java applications to communicate with a variety of other applications in a Web services environment. Both ends of the communication are not required to be Java, because this standard allows any application that supports RPC to be invoked. JAX-RPC imposes very strict standards on the implementation of the API. Any client that is compliant with JAX-RPC can interoperate with any server that is compliant with JSR-101, whether a Java server or not. In addition, any client that is compliant with JSR-101 can interface with any Java server that is compliant with JAX-RPC.

A JAX-RPC server application's entry point is also known as an endpoint. A Web service endpoint is described using a WSDL document. JAX-RPC is about Web services interoperability across heterogeneous platforms and languages, making JAX-RPC a key technology for Web services based integration.

JAX-RPC clients

A JAX-RPC client is capable of invoking a Web service irrespective of whether the service has been defined on the J2EE platform or on a non-Java platform. JAX-RPC clients do not necessarily have to be Java clients. For Java clients to be JAX-RPC compliant, they have to comply with JSR-101. JAX-RPC clients can run inside a J2EE container or as a stand-alone Java client. If running inside a J2EE container, they can be coded as managed or unmanaged clients. If running as a stand-alone Java client, they can only be unmanaged.

When you implement a JAX-RPC client, consider three essential operations:

- Instantiate the locator class. The locator class has information about the Web service. The locator class is populated based on the content of the Web service's WSDL file.
- Instantiate the service using the locator class. The service interface is a local representation of the Web service. The service interface is implemented by the so-called stub class.
- Invoke a method on the service interface.

JAX-RPC client programming styles

If the Web service is not expected to ever change, the JAX-RPC client implementation works very well. This mechanism is known as a *static stub-based invocation* of a Web service. However, if the Web service were to change, the client would have to be changed accordingly. WebSphere Application Server therefore provides the capability for clients to be dynamic. There are three types of Web services clients:

- Static stub. A static stub-based JAX-RPC client uses proxy classes to communication with the Web service. These proxy classes are generated from the WSDL of the Web service.
- Dynamic proxy. In dynamic proxy clients, the default destination of the Web service can be changed in the client by specifying a different destination in the client application.
- Dynamic invocation interface (DII). DII is used when the WSDL of the Web service can change considerably over time. DII-based clients do not use proxy classes but instead read the entire WSDL file during runtime:
  - Instantiate a DII service class
  - Instantiate a Call object (Call is a class provided by JAX-RPC)
  - Populate the Call object
  - Invoke the Web service operation on the Call object

Using Table 2-1, you can get a clear idea of when each type of implementation would be used for the client and the frequency of that implementation.

Table 2-1 Client styles

Static stub	Dynamic proxy	DII
Web service not expected to change	Some changes to the Web service expected, especially the location of the service	Considerable changes to the Web service expected such as: ► Location of the service ► Request/response format ► Data types
Most common	Less common	Least common

2.1.3 Service-oriented architecture

There is a strong trend for companies to integrate existing systems to implement IT support for business processes that cover the entire business cycle. Today, interactions exist using a variety of schemes that range from very rigid point-to-point electronic data interchange (EDI) interactions to open Web auctions. Many companies have already made some of their IT systems available to all of their divisions and departments, or even to their customers or partners on the Web. However, techniques for collaboration vary from one case to another and are thus proprietary solutions where systems collaborate without any vision or architecture.

This trend has created an increasing demand for technologies that support the connecting or sharing of resources and data in a very flexible and standardized manner. Because technologies and implementations vary across companies and even within divisions or departments, unified business processes could not be supported smoothly by technology. Integration has been developed only between units that are already aware of each other and that use the same static applications.

There is a need to further structure large applications into building blocks in order to use well-defined components within different business processes. A shift towards a service-oriented approach not only standardizes interaction but also allows for more flexibility in the process. The complete application chain within a company is divided into small, modular, functional units, also known as services.

A service-oriented architecture (SOA) has to focus on how services are described and organized to support their dynamic, automated discovery, and use. Companies and their sub-units should be able to easily provide services. Other business units can use these services to implement their business processes. This integration can be performed ideally during the runtime of the system, not just at the design time.

Requirements for an SOA

For an efficient use of a service-oriented scenario, a number of requirements have to be fulfilled:

- Interoperability between different systems and programming languages

  The most important basis for a simple integration between applications on different platforms is a communication protocol, which is available for most systems and programming languages.

- Clear and unambiguous description language

  To use a provider service, it is not only necessary to access the provider system, but also the syntax of the service interface must be clearly defined in a platform-independent fashion.

- Retrieval of the service

  To allow a convenient integration at design time or even runtime of the system, we require a mechanism that provides search facilities to retrieve suitable available services. Such services should be classified into computer-accessible, hierarchical categories, or taxonomies, based upon what the services in each category do and how they can be invoked.

- Security

  Protection of the services, including the information passed to and received from the service against unauthorized and malicious access, must be supported by the platform to win the confidence of the requestor (chain) at the end the business customer(s). The type and extent of security depends on the type and placement of the participants — service requestors and service providers — and the services themselves. Service usage monitoring and security incident action plans have to be in place to detect unauthorized access (attempts) and trigger counter measures. Even though security feels contradictory to the service idea, it is required to empower and retain authenticated and authorized requestors or customers while fencing off everything and everyone.

2.1.4 Security

Web services security is an important Web services issue. You would not think of publishing or using a business service, such as a stock purchase or bank account transfer, without a proper security framework. In WebSphere Application Server V6, Web services security can be provided at the transport level (SSL) and at the message level (WS-Security specification). Also available is a new facility, the Java Authorization Contract for Containers (JACC). JACC allows a third-party authorization provider to plug into an application server such as WebSphere Application Server using standard interfaces to make access decisions. You can architect highly secure client-server designs using these security levels.

This section addresses the main security requirements in an application that is based on Web services. It briefly covers the transport channel security solutions. Then, most of the discussion focuses on how to use fully the WS-Security specification support that is available in WebSphere Application Server V6.

J2EE Security beginnings

Prior to the J2EE 1.4 specification, security was implemented at the method level inside the web.xml deployment file and the ejb-jar.xml file for EJB method permissions. This implementation restricted access to specific methods and servlets by role within an application. When a method was accessed that had a security role associated with it, the application would prompt the user to provide credentials in the form of a user name and password. If the credentials checked successfully, a Subject was created with the user information including the groups to which the ID belongs. The application would then get the roles for the method from the deployment descriptor and verify that the roles intersected. If there was an intersection, the method was permitted. Otherwise, it would fail. WebSphere Application Server also introduced two special subjects for J2EE applications:

- AllAuthenticated. Implies that any valid user in the registry has the ability to access the method.
- Everyone. Implies that any user, authenticated or not, can access the method.

There were limitations to this model, however. All access decisions were made by the application server implementation. If a third party plug-in was preferred, it required proprietary interfaces to access. There were no standard methods for authorization providers to plug into the access decision process. There was no standard method for third-party providers to collect the security policy information from the application. To address these limitations, JACC was added to the J2EE 1.4 specification.

JACC

With JACC, you can use third-party authorization providers to plug into application servers, such as WebSphere Application Server, using standard interfaces to make decisions. This ability allows an authorization provider to handle both the J2SE and the J2EE permissions using the JACC interface. Permission classes are defined in the new standard to handle both EJB and Web permissions in a single implementation. Contract requirements are now defined in a new package, javax.security.jacc. JACC does not specify a standard on how to assign principals to roles and permissions.

Deployment tools requirements

New procedures had to be implemented to take advantage of the new interface. During an application installation, the deployment descriptor security policy must be translated to the appropriate JACC permission objects. You must then associate the permission objects with the appropriate roles. The next step is to create a unique identity (contextID) for the module being deployed. You would then propagate the information to the provider using the PolicyConfiguration object implemented by the provider. The last step is to link all the modules in an application and commit.

Application server container requirements

For the application server to use the new JACC interface, changes had to be made in the way the container used security. The container must first authenticate the user by checking credentials. The container then creates a permission object for the resource being accessed. Next, the container uses PolicyContextHandler objects to register the required information and create a unique identity for the module that is being accessed. Finally, the container calls the Policy object that the authentication provider implements to make the access decision.

2.2 Flow languages

The ability to follow specific business processes becomes important as you develop and deploy complex business applications. WebSphere Application Server allows you to create work flows that you can use to define specific business processes. This section explains the evolution of workflow creation using the flow languages that are available to WebSphere.

2.2.1 Flow Definition Language

Flow Definition Language (FDL) is the most generic description of any work flow programming methodology. By defining a work flow in a declarative rather than programmatic way, an FDL is required to specify the steps that are required to complete the workflow. When defining a flow, there are multiple trigger events that can start a workflow.

- A business event can start a workflow based on the change in state of some item in the business process, such as the receipt of a trigger from another application.
- Scheduling allows work flows to be started automatically, where no human intervention is required to begin the business process. An example of an automatic start to a work flow would be a nightly inventory update routine that determines, based on stocking levels, which items need to be ordered for the business.
- A human intervention can start a business process. For example, when an invoice is received, and someone then enters the information from the invoice into the accounting system for payment processing.

2.2.2 Flow Definition Markup Language

Process models in WebSphere Application Server prior to V6 were described using Flow Definition Markup Language (FDML). FDML is the markup language that is based on the process-related aspects of Web services Flow Language (WSFL). It is used to develop process-driven applications. Most current Integrated Development Environments (IDE) offer graphical mapping of business processes, and the markup language is never directly edited. WebSphere Studio Application Developer Integration Edition provides a wizard to aid in this implementation.

2.2.3 Business Process Execution Language

Business Process Execution Language for Web services (BPEL4WS or BPEL for short) is the J2EE 1.4 standard for Web services business process composition. This new standard is supported fully in WebSphere Application Server V6. BPEL provides for the formal specification of business processes and business interaction protocols. It extends the Web services interaction model and enables it to support business transactions. Processes in BPEL export and import functionality by using Web service interfaces exclusively. BPEL represents a convergence of the ideas in Microsoft's XLANG and IBM WSFL specifications. Both XLANG and WSFL are superseded by the BPEL4WS specification.

BPEL4WS allows you to specify business processes and how those processes relate to Web services. This includes specifying how a business process makes use of Web services to achieve its goal, as well as specifying Web services that are provided by a business process. Business processes that are specified in BPEL are fully executable and portable between BPEL-conforming environments. A BPEL business process interoperates with the Web services of its partners, whether or not these Web services are implemented based on BPEL. Finally, BPEL supports the specification of business protocols between partners and views on complex internal business processes.

Business processes can be described in two ways:

- *Executable business processe*s model actual behavior of a participant in a business interaction.
- *Business protocols* use process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior. The process descriptions for business protocols are called abstract processes.

BPEL4WS is meant to be used to model the behavior of both executable and abstract processes.

Concepts and terms in BPEL4WS

There are five major elements in the process definition.

Containers	Defines the data containers that the process uses. Provides definitions of the data containers in terms of WSDL message types. Containers allow processes to maintain state data and to process history based on messages exchanged.
Partners	Defines the different parties that interact with the business process in the course of processing the order. Each partner is characterized by a service link type and a role name. This information identifies the functionality that the business process and partner must provide for the relationship to succeed (that is, the port types that the process and the partner need to implement).
Activities	The actions that are carried out within a business process. An important action in a business process is to wait for a message to be received from a partner. This kind of action is specified using a <receive> activity. The <receive> activity specifies the partner from which the message is received, as well as the port and operation that is provided by the process and used by the partner to pass the message. The <reply> activity is used to specify a synchronous response to the request corresponding to a <receive> activity. If the response to the original request is sent asynchronously, the response is delivered using the invocation of a Web service that is provided by the requestor. Consequently, the <invoke> activity is used within the process that produces the asynchronous response. The original requestor will use a <receive> activity to process the response delivered by the <invoke> activity. A more powerful mechanism is the <pick> activity. This activity specifies a set of messages that can be received from the same or different partners. Whenever one of the specified messages is received, the <pick> activity is completed and processing of the business process continues. Additionally, you can specify that processing should continue if no message is received in a given time.

Fault handler	Contains fault handlers that define the activities that must be executed in response to faults which result from the invocation of the services. In BPEL4WS, all faults, whether internal or resulting from a service invocation, are identified by a qualified name. There is a uniform naming model for faults.
Data containers	Information is passed between the different activities in an implicit way through the sharing of globally visible data containers. This link is one of the main conceptual differences from WSFL, where data links explicitly describe how the data flows from one point to another. In BPEL4WS, this data is stored in variables that are held in a container and can be accessed by several parts of the flow. The current version of BPEL4WS supports only global data containers.

BPEL4WS runtime

The alphaWorks® Web site offers a runtime facility for BPEL. This platform:

- Executes business processes written in BPEL4WS.
- Offers a set of samples that demonstrate the use of BPEL4WS.
- Contains a tool that validates BPEL4WS documents.
- Includes an Eclipse plug-in that provides an editor for creating and modifying BPEL4WS files, supporting top-down and bottom-up approaches.

2.2.4 More information

For more information about flow languages, consult the following:

- The WSFL specification

  `http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`
- *The Web services insider: Introducing the Web services Flow Language*

  `http://www.ibm.com/developerworks/webservices/library/ws-ref4/index.html`
- *The Web services insider: Getting into the Flow*

  `http://www.ibm.com/developerworks/webservices/library/ws-ref5/`
- *Business process execution language for Web services, Version 1.0*

  `http://www.ibm.com/developerworks/webservices/library/ws-bpel1/index.html`
- *Business processes in a Web services world*

  `http://www.ibm.com/developerworks/webservices/library/ws-bpelwp/`

- IBM BPEL4WS Java runtime (BPWS4J)

  http://www.alphaworks.ibm.com/tech/bpws4j
- *XLANG— Web services for business process design*

  http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
- BPEL4WS 1.0 specification

  http://www.ibm.com/developerworks/library/ws-bpel/

2.3 J2EE Connector Architecture

The J2EE Connector Architecture (JCA) is a standard way for a Java component to connect to any enterprise system. It is part of the J2EE standard that was introduced in the J2EE 1.3 specification. WebSphere Application Server V6 is J2EE 1.4 compliant and supports the connector architecture. The JCA states that the way an application server communicates with an enterprise system (such as CICS®) is through a resource adapter.

A resource adapter is written specifically for the enterprise system that it supports. However, the same resource adapter can be plugged into any application server. Resource adapters represent a middle tier that allows Java clients to communicate with enterprise information systems (EIS) such as CICS and IMS™.

A Java client that interacts with an EIS uses a resource adapter that is specific for that EIS. For example, a client that connects to IMS requires an IMS resource adapter. The Java client uses the common client interface (CCI) API to communicate with the resource adapter. The resource adapter in turn interprets the CCI request and interacts with the EIS using a protocol that is native to the EIS. If a Java client calls a program running in CICS, the Java client uses the CCI API to send a request to the CICS ECI resource adapter, specifying the CICS program and server to call.

Resource adapters run in a J2EE application server, such as WebSphere Application Server. The application server can manage the connection between the resource adapter and the EIS, allowing the resource adapter to perform connection pooling, security propagation, and transactions.

IBM provides JCA adapters for CICS, IMS, and Host On-Demand. Other companies provide JCA adapters for a range of products, such as SAP, PeopleSoft, Siebel, and Oracle ERP. For a list of JCA adapters, refer to:

http://java.sun.com/j2ee/connector/products.html

2.4 Java Message Service

Messaging is a very popular facility for exchanging data between applications and clients of very different types. It is also an excellent tool for communication between heterogeneous platforms. WebSphere Application Server V6 recognizes the power of messaging and introduces a new messaging platform within the WebSphere environment. This messaging environment is implemented through the use of a service integration bus, which provides the framework that is needed to support SOA is an important component of the enterprise-wide bus. This enterprise bus is commonly known as the Enterprise Service Bus (ESB) or the enterprise nervous system. While JMS is a part of the default messaging provider, it is just a portion. The default messaging provider is much more than just JMS.

2.4.1 Overview of JMS

Looking at the default messaging provider from the big picture, the new facility offers significant power to applications. The environment offers integrated asynchronous capabilities for the WebSphere platform through a full JMS service provider that is JMS 1.1 compliant. The service integration bus offers an intelligent infrastructure for service-oriented integration. The service integration bus unifies SOA, messaging, message brokering, and publish/subscribe facilities. The default messaging provider does not replace WebSphere MQ, but rather it is a mechanism to share and extend messaging family capabilities.

The default messaging provider offers the following features:

- Multiple messaging patterns (APIs) and protocols for messaging and service-oriented applications.
  - A fully J2EE 1.4 compliant JMS provider that supports standard JMS APIs.
  - Implementation of relevant Web services standards, including JAX-RPC APIs.
- Reliable message transport facilities.
- Intermediary logic called mediations to adapt message flow intelligently in the network.
- Clustering enablement for scalability and high availability (HA).

- Full Integration with WebSphere Application Server.
  - A full functioning JMS 1.1 provider.
  - Full integration into server administration and runtime environment for:
    - Systems management.
    - Reliability, availability, and serviceability (RAS).
    - Security.
    - Performance monitoring infrastructure (PMI).
    - Java thread modeling and support.
  - Uses the existing installation processes.
  - Centralized management through the administrative console.
  - An all Java implementation, no external processes to manage.
- Flexible Quality of Service (QoS) implementation.
- Connectivity to existing WebSphere MQ infrastructures.
- Uses any JDBC-compliant database for the message store.

Messaging in WebSphere Application Server V6 has been improved. The default messaging provider is scalable and removes the single point of failure of the messaging server through HA facilities. The environment offers full connectivity to and co-existence with WebSphere MQ infrastructures as well as connectivity to other WebSphere Application Server V6 cells. Because the engine is now in-process, performance gains are also seen in the messaging environment. Lastly, the security is also now fully integrated.

2.4.2 JMS and WebSphere MQ

The JMS API is the standard Java API for accessing enterprise messaging systems from Java programs. In other words, it is a standard API that sends and receives applications that are written in Java. You can use these applications to access the messaging provider to create, send, receive, and read messages. This section discusses some of the important features of the JMS specification.

For a complete discussion of JMS, refer to the JMS 1.1 specification. Additionally, refer to the *WebSphere Application Server V6 - System Management and Configuration Handbook*, SG24-6451.

JMS API history

IBM, among others, was actively involved with Sun Microsystems in the process that led to the original JMS API being published in 1999. Several versions of the API have subsequently been released, with the latest being version 1.1, which includes many changes that resulted from a review of the API by the Java community.

The JMS API defines a vendor-independent programming interface. It does not define how the messaging provider should be implemented or which communication protocol should be used by clients to communicate with the messaging provider. Different vendors can produce different JMS implementations. They should all be able to run the same JMS applications, but the implementations from different vendors will not necessarily be able to communicate directly with each other.

JMS providers

JMS providers are messaging providers that provide a JMS API implementation. However, this does not mean that the underlying messaging provider is written using the Java programming language. It simply means that the JMS provider that is written by a specific vendor will be able to communicate with the corresponding messaging provider. For example, the WebSphere MQ JMS provider knows how to communicate with WebSphere MQ.

JMS domains

The JMS API introduces the concept of JMS domains. These domains are the same as the messaging models that are described in *WebSphere Application Server V6 - System Management and Configuration Handbook*, SG24-6451.

The JMS API also defines a set of domain specific interfaces that enable client applications to send and receive messages in a given domain. However, version 1.1 of the JMS specification introduces a set of domain independent interfaces, referred to as the *common interfaces*, in support of a unified messaging model. The domain specific interfaces have been retained in version 1.1 of the JMS specification for backwards compatibility.

The preferred approach for implementing JMS client applications is to use the common interfaces. Thus, the JMS code examples in this chapter use the common interfaces.

JMS administered objects

Administered objects encapsulate JMS provider-specific configuration information. They are created by an administrator and are later used at runtime by JMS clients.

The JMS specification states that the benefits of administered objects are that:

- They hide provider-specific configuration details from JMS clients.
- They abstract JMS administrative information into Java objects that are easily organized and administered from a common management console.

The JMS specification defines two types of administered objects:

- JMS connection factories

  A connection factory encapsulates the configuration information that is required to connect to a specific JMS provider. A JMS client uses a connection factory to create a connection to that JMS provider. Connection factory objects support concurrent use. That is, they can be accessed at the same time by multiple threads within a JMS client application.

- JMS destinations

  A destination encapsulates addressing information for a specific JMS provider. A JMS client uses a destination object to address a message to a specific destination on the underlying JMS provider. Destination objects support concurrent use. That is, they can be accessed at the same time by multiple threads within a JMS client application.

JMS and JNDI

At runtime, JMS clients need a mechanism by which to obtain references to the configured JMS administered objects. The JMS specification establishes the convention by which these references are obtained by looking them up in a namespace using the Java Naming and Directory Interface (JNDI) API.

The JMS specification does not define a naming policy that indicates where messaging resources should be placed in a namespace. However, if the JMS client is a J2EE application, the J2EE specification does recommend that you place messaging-related resources in a JMS subcontext.

Administrators require additional tools to create and bind the JMS-administered objects into the JNDI namespace. The JMS specification places the responsibility of providing these tools on the JMS provider.

J2EE resource references and JMS

An additional consideration in this discussion is that the JMS client application needs to know where the JMS-administered object was placed within the JNDI namespace to locate it at runtime. This requirement creates a dependency between the JMS client code and the actual runtime topology. If the JMS-administered object is moved within the JNDI namespace, the JMS client application would need to be modified, which is unacceptable.

The J2EE specification adds a level of indirection between the JMS client application and the JMS administered objects within the JNDI namespace. It defines a local JNDI namespace for each J2EE component. This local JNDI namespace can be accessed by performing lookups with names that begin with java:comp/env. When a J2EE module is being assembled, the resources referenced through the local JNDI namespace must be defined in the deployment descriptor for that module. These references are mapped by the administrator to the real JMS-administered objects in the global JNDI namespace when the application is deployed to the target operational environment. At runtime, when the JMS client performs a lookup in its local JNDI namespace, it is redirected to the actual JMS-administered object in the global namespace.

JMS connections

A JMS connection object represents the actual connection that a JMS client has to its JMS provider. The JMS specification states that a connection encapsulates an open connection with a JMS provider and that it typically represents an open TCP/IP socket between a client and a JMS provider. However, this connection is dependent on the JMS providers implementation.

Note: The creation of a connection object normally results in resources being allocated within the JMS provider itself. That is, resources are allocated outside of the process that is running the JMS client. For this reason, you must take care to close a connection when it is no longer required within the JMS client application. Invoking the close method on a connection object results in the close method being called on all of the objects created from it.

The creation of the connection object is also the point at which the JMS client authenticates itself with the JMS provider. If no credentials are specified then the identity of the user under which the JMS client is running is used. Connection objects support concurrent use. Connection factory objects are used to create instances of connection objects.

JMS sessions

A JMS session is used to create message producers and message consumers for a single JMS provider. It is created from a connection object. It is also used to define the scope of transactions. That is, it can group multiple send and receive interactions with the JMS provider into a single unit of work. However, the unit of work only spans the interactions performed by message producers or consumers created from this session object. A transacted session can complete a transaction using the commit or rollback methods of the session object. After the current transaction has been completed, a new transaction is started automatically.

Session objects do not support concurrent use. Thus, they cannot be accessed at the same time by multiple threads within a JMS client application. If a JMS client requires one thread to produce messages while another thread consumes them, the JMS specification recommends that the JMS client uses separate sessions for each thread.

JMS messages

The JMS session acts as factory for JMS messages. The JMS specification defines a logical format for the messages that can be sent to, and received from, JMS providers. Remember that the JMS specification only defines interfaces and not any implementation specifics. So, the actual physical representation of a JMS message is provider specific.

The elements that make up a JMS message are:

- Headers

  All messages support the same set of header fields. Header fields contain values that are used by both clients and providers to identify and route messages.

- Properties

  Each message contains a built-in facility to support application-defined property values. Properties provide an efficient mechanism to filter application-defined messages.

- Body

  The JMS specification defines several types of message body.

The JMS specification defines five message interface children. These child interfaces allow for various types of data to be placed into the body of the message.

Message selectors

A JMS message selector allows a JMS client to filter the messages on a destination so that it only receives the messages that it is interested in. It must be a string whose syntax is based on a subset of the SQL92 conditional expression syntax. However, the message selector expression can only reference JMS message headers and properties, not values that might be contained in the message.

If a message consumer specifies a message selector when receiving a message from a destination, only messages whose headers and properties match the selector are delivered. If the destination in question is a JMS queue, the message remains on the queue. If the destination in question is a topic, the message is never delivered to the subscriber (from the subscribers perspective, the message does not exist).

For a full description of message selectors and their syntax, refer to the JMS specification at:

`http://java.sun.com/products/jms/docs.html`

JMS message producers

The JMS session also acts as factory for JMS message producers. A JMS message producer is used to send messages to a specific destination on the JMS provider. A JMS message producer does not support concurrent use. The target destination is specified when creating the message producer. However, it is possible to pass a value of null when creating the message producer. When using a message producer created in this manner, the target destination must be specified on every invocation of the send method.

The message producer can also be used to specify certain properties of messages that it sends, such as, delivery mode, priority, and time-to-live.

JMS message consumers

The JMS session also acts as factory for message consumer objects. A JMS client uses a message consumer object to receive messages from a destination on the JMS provider. A JMS message consumer does not support concurrent use. Message consumers can operate in pull mode or push mode. The JMS specification defines message consumers for both of these modes.

- Pull mode

  A JMS client operates in pull mode simply by invoking one of the receive methods on the message consumer object. The message consumer interface exposes a variety of receive methods that allow a client to poll the destination or wait for the next message to arrive.

- Push mode

  To implement a solution that uses push mode, the JMS client must register an object that implements the javax.jms.MessageListener interface with the message consumer. With a message listener instance registered, the JMS provider delivers messages as they arrive by invoking the listener's onMessage method.

An instance of the message listener can now be registered with the JMS message consumer by the JMS client application. When the listener is registered, the connection needs to be started in order for messages to be delivered to the message listener.

JMS exception handling

Any runtime errors in a JMS application will result in a javax.jms.JMSException being thrown. The JMSException class is the root class of all JMS API exceptions.

A JMSException contains the following information:

- A provider-specific string that describes the error.
- A provider-specific string error code.
- A reference to another exception. The JMSException is usually caused by another exception being thrown in the underlying JMS provider. The JMSException class allows JMS client applications to access the initial exception using the getLinkedException method. The linked exception can then be used to determine the root cause of the problem in the JMS provider.

The implementation of JMSException does not include the embedded exception in the output of its toString method. Therefore, it is necessary to check explicitly for an embedded exception and print it out.

Note that in the JMS point-to-point domain, messages remain on a destination until they are either received by a message consumer or they expire. In the JMS publish/subscribe domain, messages remain on a destination until they have been delivered to all of the registered subscribers for the destination or until they expire. For a message to be retained when a subscribing application is not available, the subscribing application must create a durable subscription. However, when using a message listener to receive messages asynchronously, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to the receive methods on the message consumer.

The JMS API provides the javax.jms.ExceptionListener interface to solve this problem. An exception listener allows a client to be notified of a problem asynchronously. The JMS client must register an object that implements this interface with the connection using the setExceptionListener method. With an exception listener instance registered, the JMS provider invokes its onException method to notify that a problem has occurred.

Application server facilities

The JMS specification defines a number of optional facilities that are intended to be implemented by JMS providers and application server vendors. These facilities extend the functionality of JMS when the JMS client is executing within the context of a J2EE container. The application server facilities are concerned with two main areas of functionality:

- Concurrent message consumers

  Session and message consumer objects do not support being accessed from multiple threads concurrently. Such a restriction would be a huge obstacle to implementing JMS applications within an application server environment, where performance and resource usage are key concerns. The application server facilities define a mechanism that allows an application server to create message consumers that can concurrently process multiple incoming messages.

- Distributed transactions

  The JMS specification states that it does require a JMS provider to support distributed transactions. However, it also states that if a provider supplies this support, it should be done via the JTA XAResource API. The application server facilities define the interfaces that an application server should implement to correctly provide support for distributed transactions.

JMS and J2EE

The JMS API was first included in version 1.2 of the J2EE specification. This specification required that the JMS API definitions be included in a J2EE product, but the platform was not required to include an implementation of the JMS ConnectionFactory and Destination objects.

Subsequent versions of the J2EE specification have placed further requirements on application server vendors. WebSphere Application Server V6 is fully compliant with version 1.4 of the J2EE specification, which states the following with regard to the JMS API:

- A JMS provider must be included in a J2EE product. The JMS implementation must provide support for both JMS point-to-point and publish/ subscribe messaging, and thus must make those facilities available using the ConnectionFactory and Destination APIs.
- The JMS specification defines several interfaces intended for integration with an application server. Because the JMS API does not specify the architecture for pool implementation, a J2EE-compliant product does not need to provide objects that implement these interfaces. For the same reason, portable J2EE applications (those that are expected to move between application server implementations) must not use the following interfaces:
  - javax.jms.ServerSession
  - javax.jms.ServerSessionPool
  - javax.jms.ConnectionConsumer
  - all javax.jms XA interfaces

WebSphere Application Server V6 also provides full support for the application server facilities, which are described in “Application server facilities” on page 36.

2.4.3 Advantages of using JMS

JMS is a generic Java messaging API. Applications that use the API can communicate with any messaging provider software that conforms to the JMS specification.

Before JMS, client applications could only use vendor-specific APIs to communicate with that vendor's messaging software. For example, to write an application in Java that could communicate with WebSphere MQ (the messaging software from IBM), the application had to make use of the WebSphere MQ Client for Java API. Of course, an application that uses a vendor-specific API is then tied to that particular messaging software. This restriction also meant that programmers had to learn a different API for different vendor messaging software. JMS presents a standard API.

2.4.4 Disadvantages of using JMS

JMS functionality is only as good as the JMS implementation of the JMS provider. Clearly, if the provider implementation does not conform strictly to the JMS standard, or if it conforms but the implementation is poor in other respects (in terms of robustness, for example), then experiences with JMS might be less than satisfactory. There are many messaging solutions that implement JMS, including many open source solutions. One way to ensure that a provider is sound and is more likely to conform with the JMS standard is the licensing that Sun Microsystems grants. Sun, which produces the JMS specification, grants licenses to J2EE platform vendors who make a commitment to compatibility that is based on compatibility tests for JMS. IBM is such a licensee.

JMS is a standard that is attempting to strike a balance between being generic and being feature rich. The JMS 1.1 specification states:

> *If JMS provided a union of all the existing features of messaging systems it would be much too complicated for its intended users. On the other hand , JMS is more than an intersection of the messaging features common to all products.*

While JMS can give a programmer a simpler, standard API, it can also cut off access to more powerful features that are available with vendor-specific messaging software.

Some areas that JMS does not give a specification for are:

- Load balancing/fault tolerance
- An API for administering messaging products
- An API for controlling the privacy nor integrity of messages
- A wire protocol, that is a protocol for moving messages around
- A message type repository

However, because the JMS specification does not include specifications for those areas, does not mean that JMS providers cannot implement this functionality additionally in the JMS provider's platform. The default messaging provider fully implements JMS 1.1, and it also provides more functionality. Of course, these features are specific to WebSphere.

Furthermore, because there is a requirement for messaging, does not mean that you have to use JMS. It is possible to use, for example, WebSphere MQ and connect to it with the WebSphere MQ specific Java API.

You might want to consider not using JMS if some or all of the following statements are true:

- You want to connect to existing messaging infrastructure.
- You want to communicate between J2EE and non J2EE applications with messages.
- It is unlikely that applications need to be messaging software independent.
- There is vendor specific functionality that you want to use, and JMS does not support it.
- The specific messaging software provides a mature, non-JMS Java API.

2.4.5 Summary

JMS is still evolving. It provides a standard client API for messaging in Java. Each release of the specification adds useful features. You should JMS where possible. However, there are alternatives to JMS. In the short term, you might want to consider an alternative that might be more attractive, especially in complex and legacy messaging environments.

You can find more information about JMS at:

`http://java.sun.com/jms`

2.5 Business Rule Bean Framework

This section explains how to use the Business Rule Beans in building enterprise applications. For specific details about Business Rule Bean implementation refer to the *WebSphere Application Server Enterprise V5 and Programming Model Extensions WebSphere Handbook Series*, SG24-6932.

2.5.1 Planning

The Business Rule Beans framework extends the scope of the WebSphere Application Server Enterprise to support business applications that externalize their business rules. Business Rule Beans remove the volatile and rapidly changing components of a system, for example the business rules, from the application so that you can make changes to the application without touching the applications' core components, such as user interfaces, database interfaces, and business object structure.

Rule externalization is accomplished by extending the application analysis and design processes to identify the points of variability in application behavior. Business Rule Beans are implemented as standard Java components (JavaBeans or EJBs) that are managed by the WebSphere environment (much like an EJB component). To access the Business Rule Bean, a program simply needs to implement a trigger point. This trigger point interfaces with the Business Rule Bean framework to execute the business rule that is encapsulated within a Business Rule Bean. Programming a unique new rule implementation in Java is usually a simple process, made easier by the set of predefined rule implementors that can be used to create a customized business rule.

2.5.2 Business Rule Beans

A business rule is a statement that defines or constrains some aspect of a business by asserting control over some behavior of that business. A business rule officiates over frequently changing business practices and can come from government regulations, company practices, customer status, or other external factors, such as adapting business processes to react to competitive pressure. These types of rules can change periodically and do not require a rebuild of all applications that support the business process. By externalizing rule processing, the rule can change without affecting the business process. At its simplest level, a business rule is little more than a well-placed if/then statement that compares a variable against a determined value and then issues a command when they match.

2.5.3 Why use Business Rule Beans

Business Rule Beans provide a framework and a tool that facilitates externalizing business rules from the core of the business applications. Externalizing the rapidly changing laws that governs the business process allows decoupling of the rules from the process application. So, at any point in time, when performing any change to the business process rules, you do not need to rebuild the entire business application. Moreover, this separation leads to reduced maintenance and testing costs of the application code, because any rule change would require only maintaining and testing the rule itself and not the whole business application. Rule maintenance is not limited to modifying existing business rules. It also incorporates the introduction of new rules. Furthermore, changes to business rules can be made ahead of time and scheduled to take effect automatically at a specific time.

The Business Rule Beans framework provides a Rule Management tool that has a very simple administrative GUI. It can be used by business analysts who are not IT professionals to configure existing rules by changing variable business rule data. Let's say for example that an insurance company grants a 10% discount to customers who are more than 60 years old. At some point in time, the company's policy can change the discount percentage. This change can be incorporated in the system in a very simple manner. The business analyst uses the Rule Management tool and changes only the value of the discount percentage.

Business Rule Beans framework not only separates the business code from the application code but also separates user roles within an organization. In any organization applying the Business Rule Beans framework, the following three roles need to exist:

- A rule-based application component developer who provides EJBs and servlets that call out to business rules when appropriate. The component developer is conceptually unaware of the specific implementation that is going to be associated with the rule.
- A rule implementation developer who creates the Java classes that provide the concrete implementation of the business rules.
- A domain expert (business analyst) who determines the values for the business rules, such as the start and end dates for using the rule, the rule's initialization parameters, and the business intent.

This working arrangement is advantageous because these individuals are able to work in parallel. The rule implementation developer can create a rule without needing to know the values that are to be contained inside it, and the analyst can modify it knowing the rule name and the folder in which it is located using the Rule Management Application.

Business Rules can be used in virtually every application domain. Some obvious applications include personalization of Web sites, where you could configure rules to adapt the behavior of the site depending on the particular client who is accessing it. You can determine the user classification based on previous access duration to the Web site and then determine the Web site content based on the user classification. The insurance and banking industries, which are characterized by complex and mutating business requirements, can also make broad use of the rules. For example, the car insurance rate can be based on a customer's age, miles driven per month, or number of accidents within a certain duration. In banking applications, service charges can be determined based on a customer's classification as determined by the amount of investment by that customer.

2.6 Information integration

Most enterprises have data stores across multiple systems. Depending on the age of the business, some of this data could be stored in COBOL files on large mainframe systems, while other portions are on mid-range and distributed systems in different proprietary formats, or relational database management systems (RDBMS). The challenge is to provide applications to customers, partners, and internal staff that combines much of this data into a single presentable view.

Information integration builds on the solid foundation of existing data management solutions. Information integration provides an end-to-end solution for transparently managing both the volume and diversity of data that exists in enterprises and organizations today. Increasingly, business IT operations involve the need to integrate diverse and unconnected infrastructures.

The following goals are critical to increasing operations efficiency and gaining a competitive advantage:

- Integrate seamlessly with new businesses and link packaged applications with legacy systems.
- Control the accelerating costs of managing disparate systems and integrating across heterogeneous pockets of automation.
- Mitigate shortages of people and skills while quickly reaching new markets.
- Implement solutions that efficiently access and manage information across product and industry boundaries.

Information Integration patterns serve to consolidate data across multiple applications or for the entire enterprise. This consolidation is done at a level prior to the access of the data by a specific application or user. Information Integration works on the premise that the data is split across multiple stores. Some of these stores cam be structured and some unstructured. It is even quite likely that some structured data is in a different structure than other data would be. If the enterprise can provide a proven and reliable pattern for data combination of these disparate elements, you can write applications to take advantage of the unified data view and develop API that you can use within the enterprise to access the data.

This section discusses some of the approaches and trends in developing Information Integration patterns.

2.6.1 Data consolidation

Data consolidation (also know as data placement or data aggregation) is the more traditional integration approach. It involves moving the information from its various locations and storage formats, consolidating the information locally often through transformation technologies, and then loading the information into the single view.

This approach has its advantages. The extraction of data and its transformation and storage to a local copy are all done ahead of time of the actual user query. Therefore, there is less processing required when the actual user query is made, allowing for a better performance.

However, because the final querying is done on a local copy, latency is an issue. How up to date is the information? Because the majority of applications continue to access this data from the original location, issues with data currency creep up quickly. Data consolidation systems must specify how up-to-date the data in the local copy needs to be. In some situations, relatively high latency might not be a problem, where as in others, latency might need to be kept to a minimum. For example, some data can be slightly older information because the likelihood it has changed is low. Items like a phone number or a Social Security Number are not likely to change often, and do not necessarily need to be updated as frequently. Many questions arise in respect to latency and data currency. How does the new central store process updates? Are they handled as individual updates? Batch updates? How often are updates processed? These questions must be answered when planning for implementing data aggregation.

Another disadvantage in this approach is that because there are remote and local copies of the data, data consolidation does require more storage than other approaches.

Implementing a data consolidation approach can also be quite expensive for the enterprise, because applications must be written to collect the data from the different sources and then to insert that data into the new structure.

2.6.2 Distributed access

Distributed access (also known as data access or federation) is a newer approach. Distributed access again allows the user to see the single view of data, but the view is constructed from data in its original locations. In its simplest form, two relational databases have local queries performed on them, and the results are then combined in a federated final query that joins the two tables that result from the local queries.

The advantage of this approach is that the data is being extracted from its original locations. Thus, latency is not an issue. The information is real-time information.

However, because the data is extracted in real time it has to be processed in real-time. Data consolidation and distributed access approaches both have to map, transform, and present data. For example, if data exists in an Oracle database, a DB2 database, and a mainframe COBOL file, then the integration system has to be concerned with extracting the relational data for joining, must be aware of and handle the specifics of the two RDBMS, and then handle the integration of the non-RDBMS data from the mainframe.

In the data consolidation approach, such actions are done ahead of time and locally. In the distributed access approach, all the processing is done at the time of the user query and in various locations. Therefore, performance in a distributed access approach can be affected. You can use caching to help performance in distributed access systems. However, doing so can raise questions of latency that the distributed access approach is attempting to avoid.

Distributed access sounds very flexible. Surely the user is in a position to make unplanned complex queries of the data? In reality, a distributed access system must be extremely well planned in the combinations of queries that it will and will not handle. Otherwise, in a poorly planned distributed access system, the data that is extracted could be misleading and performance during extraction could be affected.

2.6.3 Choosing and combining data integration approaches

It is clear then that the data consolidation and distributed access approaches to data integration have their strengths and weaknesses. Each approach is better for certain situations.

Data consolidation is generally better when:

- Read-only access to reasonably stable data is required.
- Users need historical or trend data.
- Performance and availability are the priority.
- Users' needs are repeatable and can be predicted in advance.
- Transformations or joins involved are complex or long-running.

Distributed access is generally the better choice when:

- Real-time or near to real-time access is required to rapidly changing data.
- Direct immediate write access to the original data is required.
- It is technically difficult to use copies of the source data.
- The cost of copying data exceeds that of accessing it remotely.
- It is illegal of forbidden to make copies of the data.
- Users' needs are not known in advance.

The two approaches are, therefore, complimentary because they do not satisfy all requirements for all situations on their own. A data integration strategy might use each approach for different problems. Further more, each approach can make use of the other approach. That is, not only can a data integration system make use of the two approaches in parallel, but it is also possible for each approach to use the other approach within itself. Federated queries can use data consolidation techniques and consolidated data approaches can use federated queries behind the scenes. Clearly, trade offs in terms of data currency and performance are involved as a result.

2.6.4 Data integration tools

A new trend in the market is to implement a tool that does the data integration for you. IBM offers a number of solutions that fall into the data consolidation and distributed access approach categories.

Distributed access tools

The following are IBM offerings that support distributed access data integration.

- DB2 Information Integrator

  The DB2 Information Integrator (and its predecessor products IBM DB2 DataJoiner® and IBM DB2 Relational Connect) provide for the distributed access approach.

  Previously, products like IBM DB2 offered a method of federating disparate databases to look like one database. Then, using mappings in the product, the administrator was able to create views that spanned any of the databases in the federation. This, however, was somewhat limited in scope, because it required the data be accessible through some RDBMS on the back end of the federation.

  DB2 Information Integrator, removes that restriction. At first, the product supported XML documents, spreadsheets, message queues, Web content, Web services, and file systems. The tool now also allows users to map data in VSAM, IBM IMS, CA-IDMS, CA-Datacom, and Adabas. The product also touts Partner Offerings that will provide interfaces to other formats in the future.

The premise is very similar to that described by classic data federation. An Information Integrator node is logically placed in front of the data sources. Using the provided mapping tools, it creates a single point view of the data. This view is then accessible over standard J2EE API for WebSphere Application Server based applications, or stand alone Java applications. The API support includes EJB, Web services, JCA, and JMS. In addition, the DB2 Information Integrator offers an API environment to write custom wrappers and functions not normally provided with the base application, which can further enhance the functionality of non-traditional data storage.

DB2 Information Integrator supports:

- DB2, Informix, Oracle, Sybase, Microsoft SQL Server, and Terradata databases as well as ODBC sources.
- XML, Microsoft Excel, OLE DB, and flat files.
- Web services, messaging queues, and application data sources.
- IBM Lotus Extended Search sources (content repositories LDAP directories, Web, email databases, Syndicated content, and so forth).

- DB2 Information Integrator for Content

  This offering also provides for distributed access data integration but tailored toward content integration. DB2 Information Integrator for Content is a successor to IBM Enterprise Information Portal. This offering offers sophisticated searching and data mining functionality.

Data consolidation tool

IBM offers DB2 Warehouse Manager that supports data consolidation data integration. This software offering is primarily for DB2 datasource consolidation. It supports SQL, Web services, and a rich pre-built set of transformations for source to target transformations. It can use DB2 Information Integrator as a source. This software is an example of a data consolidation technology that uses distributed access technology behind the scenes. An IBM Business Partner, Ascential Software, provides a product called Ascential DataStage which provides support for heterogeneous datasources.

Replication tools

Replication minimize the latency between distributed datasource nodes and a central store as used in data collaboration. There are two IBM products that handle replication for RDBMS:

- DB2 DataPropogator, which provides replication between DB2 family products.
- DB2 Information Integrator, which provides replication among mixed RDBMS.

2.6.5 Summary on data integration

This section has shown that there are two major approaches to Data Integration. These approaches are complimentary approaches. The short comings of each approach are compensated by the advantages of the other approach. Because they are complimentary, the approaches are often mixed at a macro and micro level in data integration solutions.

Implementing data integration solutions can be very complex and expensive. IBM and Ibm Business Partners offer a rich set of software tools for implementing data integration solutions. These solutions solve many common data integration problems and therefore reduce complexity and cost.

For more information about integration visit the following sites:

- DB2 Information Integration site

  `http://www.ibm.com/software/data/integration`

- Information integration, distributed access, and data consolidation white paper

  `ftp://ftp.software.ibm.com/software/data/pubs/papers/etl.pdf`

What’s new?

This chapter provides an overview of the new features of WebSphere Application Server V6 and contains the following sections:

- Features comparison between versions
- Installation improvements
- Administration
- Default messaging provider
- Clustering enhancements
- Security enhancements
- Deprecated features in this version
- Application programming model

3.1 Features comparison between versions

The following table illustrates a features comparison between the different configurations in WebSphere Application Server V5 and WebSphere Application Server V6.

Table 3-1 Features comparison across configurations and versions

Features	**WebSphere Application Server Express V5**	**WebSphere Application Server V5**	**WebSphere Application Server - Express V6 and WebSphere Application Server V6**	**WebSphere Application Server Network Deployment V6**
J2EE application supported levels	J2EE 1.3 but no EJB nor JCA support	J2EE 1.2 and 1.3	J2EE 1.2, 1.3 and 1.4	J2EE 1.2, 1.3 and 1.4
Web services support	Only for JavaBean	EJB and JavaBean	EJB and JavaBean	EJB and JavaBean
UDDI	No	No	Yes	Yes
Web Services Gateway	No	No	No	Yes
Tooling	WebSphere Studio Sites Developer V5.x	Application Server Toolkit	Rational Web Developer (Express only), and Rational Application Developer Trial (for WebSphere Application Server) and Application Server Toolkit (for both)	Application Server Toolkit and Rational Application Developer Trial
Programming Model Extensions	No	No	Yes	Yes
Work Load Management and failover	No	No	No	Yes
Embedded JMS Server support	No	Yes	Yes - part of a bigger platform messaging infrastructure	Yes - part of a bigger platform messaging infrastructure

3.2 Installation improvements

The biggest improvement to installation is that now you can install one copy of the core files (binary WebSphere Application Server files) on a machine and then use **profiles** to define multiple application server runtime environments that share the core files. This improvement has many positive ramifications for preparing, installing, maintaining, and removing installations, including a decreased footprint.

This new functionality provides the chance to have multiple WebSphere Application Server V6 copies installed and running on the same physical machine, using different installation directory paths and different TCP/IP ports in the case of those multiples installations running at the same time. This functionality enables the system administrator to have, in a single system, different copies of the same product at different maintenance levels (namely different fixpack levels).

In addition, for each of those installations, you can create multiple profiles to share the product binary files but have different instances called WebSphere Profiles. These WebSphere Profiles behave as separated, individual installations. The custom data for each WebSphere Profile is stored in a separate directory.

Another big change in the installation process is the introduction of separate installation routines for the application server product, the Web server, the Web server plug-ins, and the clients so that you can install only what you need on a particular machine.

3.2.1 Redesigned launchpad

A redesigned launchpad on the product CD launches one of the installation programs. The launchpad also provides product solution roadmap diagrams to let you decide as you go what you want to install and where you want to install it.

3.2.2 Granular installation

New, more granular installation procedures simplify installation and improve installation reliability. Installation routines exist for the following components on the product disc:

- WebSphere Application Server products
- IBM HTTP Server
- Web server plug-ins
- Application clients

Separate installation procedures for the IBM HTTP Server product and for the Web server plug-ins let you install only what you need on a particular machine.

3.2.3 WebSphere profiles

WebSphere Application Server V5 provided the ability to set up and run multiple instances (WebSphere instances) on a single machine using a single IBM WebSphere Application Server installation. WebSphere Application Server V6 provides the WebSphere profiles. These are the WebSphere instances enhanced.

WebSphere product files have two components:

- A set of shared read-only product static files or product binaries to be shared by any functional instance of the WebSphere Application Server product.
- A set of user data set of configurable files that are customized. This set of files are called WebSphere profiles. User data includes WebSphere Application Server configuration, applications, properties, and so on.

Each profile uses a separate configuration but uses a single installation of the WebSphere binaries. Each profile is distinguished by its base path, its own directory structure, and its own setupCmdLine script to configure its command-line environment.

WebSphere profile support provides:

- The ability to create different user data (WebSphere Profiles) and run multiple instances of WebSphere Application Server sharing the product binaries.
- WebSphere profiles that define a functional instance of the server.
- The ability to have multiple profiles running of the same shared product binaries.
- Profile templates that you can use to create different server profiles or instances.
  - Stand-alone Node, Deployment Manager, Managed Node.
  - Advanced users can tweak existing profile templates to create new profile instances.
- The default application server or Deployment Manager are WebSphere profiles that are created as part of the product install process.
- The **`wsProfile`** command line tool that you can use to create additional WebSphere profiles based on templates.
- WSProfile, a Java API for profile management, that is used by **`wsProfile`**.
- A Profile Creation Tool.

3.2.4 Installing WebSphere Application Server products

WebSphere Application Server V6 is an integrated platform that contains an application server, Web development tools, a Web server, and additional supporting software and documentation.

With WebSphere Application Server - Express V6 and WebSphere Application Server V6, the launchpad installs the core product files and a stand-alone application server in a profile named default.

With WebSphere Application Server Network Deployment V6, the installation is a two-step process. The first step is using the installation wizard to install a shared set of core product files. The second step is using the new profile creation wizard to create a Deployment Manager profile, an Application Server profile, or a Custom profile.

WebSphere Application Server V6 provides a set of profile templates to create different profile types. These templates are part of the product files. Each template is a set of documents and actions where documents define the initial starting point for the profile directory and files. Actions, on the other hand, define what to do after the documents have been laid down for a profile.

Based on the profile types and templates, the administrator can create multiple profile instances of any profile. Each profile instance is registered in a registry XML file called Profile registry. The profile registry contains information and commands for specific profile instances.

3.2.5 Installing IBM HTTP Server

The installation wizard for IBM HTTP Server V6 has been completely redesigned in this version. At the end of the Web server installation, launch the new plug-in installation wizard to configure the IBM HTTP Server and the WebSphere Application Server.

3.2.6 Installing application server clients

A client application processes on a client machine and a host WebSphere Application Server machine. A client might provide the GUI, but it processes data on the host, for example. The client's installation wizard installs environments for running client applications on the client machine. Some environments perform all necessary handshaking and protocol. Thin client environments require client applications to have their own protocols, such as JNDI lookups.

3.2.7 Improved directory structures

The default installation root varies from one operating system to another but now includes IBM in the path. The installation root is the same for all three product editions now. There is no longer a separate installation root for WebSphere Application Server Network Deployment.

With the introduction of the WebSphere profiles, there are major changes in how the directories are structured. WebSphere Application Server V6 binary files are separated from installation-specific configuration information (WebSphere profiles), leading to easier installation, maintenance, and configuration management.

The V6 product installation program places the files that it creates into one of two separate environments. It installs the binary system files that might be read-only. It also creates an initial profile, which is a run-time execution environment that includes configuration files, the default location for deployed applications, logs, and other data. All profiles on a machine can share the same system files but do not change the system files.

3.2.8 Reduced footprint

With multiple WebSphere profiles in individual directories, you can have multiple stand-alone application servers that share the binaries from one WebSphere Application Server V6 installation. This environment would be a single installation with multiple profiles and is covered in Chapter 9, “Topologies” on page 247.

3.2.9 Removal and reinstallation

With V6, removal and reinstallation of application server environments is simplified and faster. Application server environments are easier to install and to uninstall. Uninstalling an application server does not change the shared system files of the base WebSphere Application Server product. WebSphere Application Server V6 does not require a clean machine when you uninstall and reinstall. Application servers uninstall cleanly and almost effortlessly.

3.3 Administration

There are changes in the administration for WebSphere Application Server V6, starting with a new URL and port to the administrative console. You can now access the administrative console at:

```
http://<fully.qualified.hostname>:9060/ibm/console
```

3.3.1 Application deployment

Deploying applications is significantly easier and more efficient with V6. Highlights include JSR-088 support, simplified EAR file configuration, administrative support to help you migrate your WebSphere Application Server V5 applications, and the ability to perform fine-grained application updates.

When updating an application, only the portion of the application code that actually changed needs to be presented to the system. The application management logic calculates the minimum actions that the system needs to execute to update the application. Under certain circumstances, the update can occur without stopping any portion of the running application.

WebSphere Application Server V6 introduces an administrative console wizard that can update deployed applications or modules in the following ways:

- Replace an entire application (.ear file).
- Replace, add, or remove a single module (.war, EJB .jar, or connector .rar files).
- Replace, add, or remove a single file.
- Replace, add or remove multiple files by uploading a compressed file.

If the application is updated while it is running, WebSphere Application Server automatically stops the application or its affected components as needed, updates the application logic, and restarts the stopped application or its components.

A new action, rollout update, sequentially updates an application that is installed on multiple cluster members across a cluster. After you update an application's files or configuration, use the rollout update option on the administrative console enterprise applications page to install the application's updated files or configuration on all cluster members of a cluster on which the application is installed.

The rollout update option does the following for each cluster member in sequence:

- Saves the updated application configuration.
- Stops all cluster members on a given node.
- Updates the application on the node by synchronizing the configuration.
- Restarts the stopped cluster members on that node.

This action updates an application on multiple cluster members while providing continuous availability of the application.

As part of the JSR-088 support, you can install J2EE modules on an application server that WebSphere Application Server provides using the J2EE Deployment API Specification (JSR-088). JSR-088 defines standard APIs to enable deployment of J2EE applications and stand-alone modules to J2EE product platforms. WebSphere Application Server is a J2EE 1.4 specification-compliant platform that implements the JSR-088 APIs.

JSR-088 defines a contract between a tool provider and a platform that enables tools from multiple vendors to configure, deploy and manage applications on any J2EE product platform. The tool provider typically supplies software tools and an integrated development environment for developing and assembly of J2EE application modules. The J2EE platform provides application management functions that deploy, undeploy, start, stop, and otherwise manage J2EE applications.

You can find further details about the Java Specification Requests at the following:

http://www.jcp.org/en/jsr/all

Finally, there is a new, simplified EAR file configuration. The companion Application Server Toolkit enables you to define the required configuration, such as a data source, as a part of the application. At deployment, you can choose to process the embedded configuration data, which automatically sets up the required configuration for the application.

3.3.2 Incremental cell version upgrade

The WebSphere Application Server V6 Deployment Manager can manage both WebSphere Application Server V6 and V5 or later nodes. This allows cells to be upgraded to a new release one node at a time, with minimal impact to the applications that are running within the cell.

When upgrading to WebSphere Application Server V6, there is no need to export the cell configuration and recreate it in a new cell. However, there are limitations and restrictions on the node, server, cluster, and application operations in a mixed node environment.

This ability of WebSphere Application Server V6 Deployment Manager to manage both versions is also extensive to administrative clients. When displaying or modifying the properties for a node, node agent, or server, the administrative console is aware of the version of the node on which the object resides and only displays those properties that are valid for that version.

3.3.3 WebSphere configuration archive

The WebSphere configuration archive is a complete set or subset of WebSphere Application Server configuration archives. Basically, it is the same as the real-time WebSphere Application Server configuration except that it might be partial. The configuration information is virtualized to make it portable (for example, it removes any installation specific information such as host name).

The WebSphere configuration archive is used to import or export configurations into or from WebSphere Application Server. It allows a declarative way to modify application server configurations.

3.3.4 Enhanced administrative infrastructure

The J2EE 1.4 specification added several requirements for application server vendors to implement in support of administration. This revision of the J2EE specification adds requirements to support:

- Java Management Extensions
- J2EE Management Specification
- J2EE Connector Architecture

In addition to J2EE specification related administration features, the embedded messaging component of WebSphere Application Server that supports Java Message Service (JMS) has been redesigned to be better integrated with the application server administration.

Java Management Extensions

Java Management Extensions (JMX 1.2) provides a standard way of exposing Java resources, for example application servers, to a system management infrastructure. Using the JMX framework, a provider can implement functions, such as listing the configuration settings and editing the settings. This framework also includes a notification layer that management applications can use to monitor events such as the startup of an application server.

The WebSphere Application Server V6 implementation of JMX includes:

- All processes that run the JMX agent.
- All run-time administration that is performed through JMX operations.
- Connectors that are used to connect a JMX agent to a remote JMX-enabled management application. The following connectors are supported:
  - SOAP JMX Connector
  - Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP) JMX Connector

- Protocol adapters that provide a management view of the JMX agent through a given protocol. Management applications that connect to a protocol adapter are usually specific to a given protocol.
- The ability to query and update the configuration settings of a run-time object.
- The ability to load, initialize, change, and monitor application components and resources during run-time.

Each Java Virtual Machine (JVM) in WebSphere Application Server includes an embedded implementation of JMX. In WebSphere Application Server V6, the JVMs contain an implementation of the JMX 1.2 specification. The JMX 1.2 specification used in WebSphere Application Server V6 is the open source MX4J package. The change to the JMX 1.2 specification does not affect the behavior of the JMX MBeans in the application server. No application server administrative APIs are altered.

The JMX V1.2 specification is backward compatible with the JMX 1.0 specification that JVMs in WebSphere Application Server V5 and later used. However, you might need to migrate custom MBeans that are supplied by products other than the WebSphere Application Server V5 to V6. The primary concern for these custom MBeans is related to the values that are used in key properties of the JMX ObjectName class for the MBean. The open source MX4J implementation more stringently enforces property validation according to the JMX 1.2 specification. You should test the custom MBeans that you deployed in previous versions of WebSphere Application Server in WebSphere Application Server V6, to ensure compatibility.

You can find details of the changes to the JMX V1.2 specification from the JMX V1.0 specification in the JMX 1.2 specification at:

`http://java.sun.com/products/JavaManagement`

J2EE Management Specification

The management layer of the JMX architecture uses an implementation of the distributed services specification (JSR-077), which is not yet part of the J2EE specification. The management layer defines how external management applications can interact with the underlying layers in terms of protocols, APIs, and so on.

J2EE Connector Architecture

WebSphere Application Server V6 supports the J2EE Connector Architecture (JCA) 1.5 specification which provides new features, such as the inbound resource adapter.

3.3.5 New administrative commands

WebSphere Application Server V6 introduces a new wsadmin scripting object, AdminTask. Various AdminTask commands are implemented for important administrative scenarios such as server management, cluster management, and resource management. AdminTask commands provide various user friendly and task-oriented wsadmin commands. AdminTask commands can have multiple steps for some complicated administrative tasks similar to the wizard in the administrative console. AdminTask commands are grouped based on their function areas.

Detailed help information for AdminTask commands and command groups is available through various AdminTask help commands. All AdminTask commands can be executed in interactive mode, which leads you step-by-step, interactively. At the end of execution, the corresponding AdminTask command is generated to help you learn the AdminTask command syntax.

3.3.6 Improved monitoring and performance tuning

There are improvements in the startup speed for applications and application servers. Two new settings enables you to fine-tune the startup speed of applications that are configured to start automatically when the server starts. A new starting weight setting lets you specify the order in which applications start when the server starts. A new background application setting lets you specify whether an application must initialize fully before its server starts or if the server can proceed without waiting for the application.

There are also improvements for monitoring the applications flow. Now, you can be more selective about what instrumentation is enabled. For example, if you want instrumentation data only for the Web container and JMS, select these data in the administrative console. Then, the detailed instrumentation data are generated only for the components you select. The edge transactions are traced for the other components that are not specified for instrumentation.

You can use filters to determine the transactions for which to record data. Request Metrics, which allows you to trace response times for individual transactions through WebSphere Application Server, provides more instrumentation and filters, including asynchronous beans, Web Services, and messaging resources. Request Metrics also supports IPv6 addressing. If a request originates from a client with IPv6 only address, filtering can be done based on the IPv6 address. Furthermore, if the server uses IPV6 addresses preferentially, the same appears in the correlators that are generated by request metrics.

The Performance Monitoring Infrastructure (PMI), which allows you to monitor the overall health of WebSphere Application Server, is now enabled out-of-the box. These monitoring APIs follow the J2EE 1.4 Performance Data Framework specification. You can enable statistics using predefined statistic sets, or you can select statistics using the custom option. With the custom option, you can enable and disable individual statistics. New additional PMI statistics, such as messaging, are provided. You also can add your own statistics using the PMI custom API.

Tivoli Performance Viewer, which provides graphical and chart views to analyze data from the PMI, is integrated into the administrative console to provide an easy, accessible, lightweight monitoring solution.

3.3.7 System applications

System applications are J2EE enterprise applications that are central to a WebSphere Application Server V6 product, such as the administrative console and file transfer application. System applications are no longer shown in the list of installed applications on the console Enterprise Applications page, to prevent users from accidentally stopping, updating, or removing them.

Because system applications are an important part of a WebSphere Application Server V6 product, system applications are deployed when the product is installed and are updated only through a product fix or upgrade. Users cannot change the metadata for system applications, such as J2EE bindings or J2EE extensions. Metadata requiring a change must be updated through a product fix or upgrade.

3.3.8 Replication

There is a new type of replication domain that is based on the new high availability framework. By using data replication domains, you do not have to configure local, remote, and alternate replicators. Any replication domains that were created with a previous version of WebSphere Application Server might be multi-broker domains.

Existing multi-broker domains remain functional. However, after you upgrade your Deployment Manager, you can create only data replication domains in the administrative console. For improved performance and easier configuration, migrate any existing multi-broker domains to the new data replication domains.

3.3.9 High availability manager

The new high availability manager function is used to eliminate (or keep to a minimum) single points of failure. The high availability manager is responsible for running key services on available application servers rather than on a dedicated one (such as the Deployment Manager). It takes advantage of fault tolerant storage technologies such as Network Attached Storage (NAS), which significantly lowers the cost and complexity of high availability configurations. It also offers hot standby and peer failover for critical services.

The high availability manager monitors the application server environment and provides peer-to-peer failover of application server components that are part of a core group. If an application server component fails, the high availability manager takes over the in flight and in doubt work for the failed server. This action significantly improves application server uptime. In a carefully planned environment, it can provide a 99.999% application server availability expectancy rate.

All components of an environment managed by the high availability manager are members of a core group. WebSphere Application Server V6 provides a default core group that is created when the product is installed. When new server instances are created, they are added automatically to this default core group. Additional core groups can be created. However, for most environments, one core group is usually sufficient, and additional core groups should not be added unless there is a specific need to do so. If your environment has more than one core group, use access point groups to enable the core groups to communicate with each other. The core groups that communicate can be in the same cell or in different cells.

3.3.10 Common networking services

The channel framework model provides a common networking service for all components, including IBM service integration technologies, WebSphere Secure Caching Proxy, and the high availability manager core group bridge service. This model consists of network protocol stacks or transport chains that are used for I/O operations within an application server environment. Transport chains consists of one or more types of channels, each of which supports a different type of I/O protocol, such as TCP, DCS, or HTTP. Network ports can be shared among all of the channels within a chain. The channel framework function automatically distributes a request arriving on that port to the correct I/O protocol channel for processing.

3.3.11 Centralized Web server plug-in configuration

The Web server plug-ins that are used to forward HTTP requests from a supported Web server to an application server can now be configured from the administrative console. Previously the configuration file for these plug-ins had to be manually edited whenever configuration changes were required.

3.3.12 Node groups

WebSphere Application Server V6 integrates an optional feature called node groups, which define a boundary for server cluster formation. Using node groups, you can cluster resources, and applications that use those resources, into a distinct partition within a cell. During the application install process, you can enable the optional resource scope validation, which notifies you if you have assigned inaccessible J2EE resources to an application.

3.3.13 Server templates

Server management functionality is enhanced in this release. This release introduces the server template functionality. You can create customized server templates that are based on an existing server configuration and use these customized templates to create new servers. This customization provides a mechanism to propagate the server configuration within the same cell easily. Furthermore, you can propagate the server configuration across the cell boundary by exporting a server configuration to a configuration archive and then importing the archive to another cell.

WebSphere Application Server V6 also introduces two new server types in addition to the existing application server:

- Generic server, which represents a generic Java or non Java server process.
- Web server, which represents an IBM or non-IBM Web server.

3.3.14 Resource providers

All functions are available for all scopes, including WebSphere Application Server V5.x and V6 nodes, for the following J2EE resources that are used by applications:

- JDBC providers
- Generic JMS providers
- WebSphere embedded JMS providers
- WebSphere MQ JMS providers
- Mail providers
- Resource environment providers
- URL providers

The generic JMS provider is available for cell scope and WebSphere Application Server V6 nodes for backwards compatibility. For WebSphere Application Server V6 nodes, you are encouraged to use the WebSphere default messaging provider.

The following J2EE resources have limitations on WebSphere Application Server V5.0 or later nodes:

- Resource adapters

  The format of resource adapter configuration has been changed considerably to accommodate JCA 1.5 in J2EE 1.4. Both JCA 1.5 and JCA 1.0 resource adapters can be defined at the cell scope. However, JCA 1.5 adapters will not be available on a WebSphere Application Server V5 or later node. For WebSphere Application Server V6 nodes and servers, both JCA 1.5 and JCA 1.0 resources can be defined. For WebSphere Application Server V5 or later nodes and servers, only JCA 1.0 resource adapters can be defined.

- WebSphere default message provider

  The WebSphere default message provider is new in WebSphere Application Server V6. Its definitions can be created at the cell scope containing WebSphere Application Server V6 and V5 or later nodes. However, it will not be available on the WebSphere Application Server V5 or later nodes. Its definitions can be created on any WebSphere Application Server V6 nodes or servers but not on a WebSphere Application Server V5 or later node or server.

3.3.15 SOAP connector

Cross release connector interoperation is currently supported only by the SOAP connector. It is unsupported for RMI connector. A WebSphere Application Server V5 or later wsadmin client can only use the SOAP connector to connect to WebSphere Application Server V6 Deployment Manager. A WebSphere Application Server V6 wsadmin client can only use the SOAP connector to connect to WebSphere Application Server V5 or later node agent or application server.

3.3.16 WebSphere profiles

Depending on the WebSphere Application Server feature is used, you need to establish which WebSphere profiles you are going to create on each server or if just a stand-alone installation is needed.

The concept of WebSphere Profile was introduced in WebSphere Application Server V5 and was called WebSphere Instance. Each WebSphere Profile uses separate configuration files but shares the binaries from which it was created.

Each profile is distinguished by its base path, its own directory structure, and its own setupCmdLine script to configure its command-line environment.

System administration commands are now profile-specific. In previous releases, commands were executed from the bin directory of the product installation root. Now commands must be executed from a particular profile/bin directory.

A given configuration can be also propagated from one WebSphere Profile to another by exporting that profile to a configuration archive and importing it to another WebSphere Profile. This mechanism works between WebSphere profiles of the same or different installations. A restriction is that this mechanism only works for an unfederated WebSphere Profiles.

3.3.17 New administrative console look and feel

The new administrative console provides an improved look and feel that is consistent across IBM products and is based on Integrated Solutions Console. It provides adapt-a-view support that allows you to change console views based on context. It also integrates the administration for IBM HTTP Server V6 and Tivoli Performance Viewer.

3.4 Default messaging provider

The embedded messaging server that shipped with WebSphere Application Server V5 was not integrated with WebSphere Application Server. WebSphere Application Server V6 includes a default messaging provider that is installed and runs as part of the product and needs no further administration. The default messaging provider is based on service integration technologies.

The default messaging provider supports JMS 1.1 domain-independent interfaces (sometimes referred to as *unified* or *common* interfaces). This new implementation enables applications to use the same, common, interfaces for both point-to-point and publish/subscribe messaging. The default messaging provider also enables both point-to-point and publish/subscribe messaging within the same transaction.

This default messaging provider integrated with the WebSphere Application Server V6 embraces:

- WebSphere Security.
- Common install process.
- WebSphere System Management. The administrative console provides MQ-Explorer type management.

- All Java implementation within the server process with no external processes. Co-exists with WebSphere MQ.
- Performance monitoring, trace, and problem determination.

Other enhancements also include:

- Clustering enablement for scalability and HA.
- More flexible Quality of Service (QoS) for message delivery.
- Connectivity into a WebSphere MQ network.
- Improved performance for in-process messaging.

3.5 Clustering enhancements

As part of the clustering enhancements, clusters in WebSphere Application Server V6 are easier to use and more consistent across the different protocols, clusters provide improved high availability characteristics.

3.5.1 Unified cluster framework

Prior to having the unified cluster framework, the routing of each protocol was done by separate groups based on specific requirements for a given protocol (cluster). The logic for HTTP servers, Web Services Gateway, and EJB servers was all unique.

The unified cluster framework standardizes how the cluster data is collected, propagated, and routed using a standard consistent architecture. In addition, as new technologies are introduced into WebSphere, they will also be able to take advantage of the framework. For example, the HTTP server's plug-in logic was different from the EJB (IIOP) routing logic which was different than the Proxy

The quality of the code has been improved because the logic is based on a single consistent architecture instead of several specific ones.

The unified cluster framework provides a more consistent way to view, configure, and administer the different types of clusters (JMS, HTTP, Web Services Gateway, IIOP, and so forth) because these clusters use the same architecture.

Clustered protocols are now able to take advantage of the high availability manager which improves the QoS.

The following is a list summarizing the clustering enhancements:

- The Edge function is integrated in WebSphere Application Server V6 with the support of unified clusters.
- The view and use of clusters is administered in a unified and consistent manner for all protocols (HTTP, EJB, JMS, JCA, and so forth).
- Third-party integration allows you better control of cluster definition when it is needed and still fits into the consistent WebSphere view.
- Continuous availability builds the infrastructure that is needed to enable a better story for doing version upgrade on continuous running systems.
- High availability makes Work Load Manager a highly available service, using the high availability manager, which makes cluster and routing information always available.
- New Work Load Manager functions (weighted distribution, eWLM integration, SLA, hardware provisioning, and so forth) are implemented once for all protocols.
- Central control of Web Container clusters.
- Backup server lists for EJBs.
- Real-time updates of the HTTP route table.
- Supports future capabilities like queuing, classification, and work prioritization in enhanced products, such as WebSphere Application Server Extended Deployment (XD).

3.5.2 Data replication service enhancements

There are a number of enhancements in WebSphere Application Server V6 that are related to the data replication service. One improvement is the integration with the administrative console, which provides:

- Improved performance and scalability.
  - Provides a more optimized communication stack.
  - Allows for use of both unicast and multicast IP.
  - Improves in the range of 4 to 8 times.
- Improves high availability and failure recovery.
  - Leverages the failure detection that is provided by high availability services.
  - Along with the Work Load Manager and Unified Clustering integration, allows for active failure recovery. For example, with HTTP Session

replication, if the affinity server for a HTTP Session goes down, Work Load Manager can route to another server that has a backup copy ready to use.

- Improves usability by leveraging group services to simplify partitioning. Has *n-replica*, where you simply define the number of backup copies that you want.

In addition, Stateful Session Beans state is also replicated.

3.6 Security enhancements

WebSphere Application Server V6 provides security infrastructure and mechanisms to protect sensitive J2EE and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability. This section discusses the set of supported specifications that are available with WebSphere Application Server V6.

3.6.1 JACC provider support

The Java Authorization Contract for Containers (JACC) specification version 1.0, introduced in WebSphere Application Server V6 and defined by J2EE Version 1.4, defines a contract between J2EE containers and external authorization providers. Based on this specification, WebSphere Application Server enables you to plug in an external provider to make authorization decisions when you are accessing a J2EE resource. When you use this feature, WebSphere Application Server supports Tivoli Access Manager as the default JACC provider.

3.6.2 Java 2 security manager

WebSphere Application Server V6 provides you with greater control over the permission granted to applications for manipulating non-system threads. You can permit applications to manipulate non-system threads using the was.policy file. However, by default, these thread control permissions are disabled.

3.6.3 J2EE Connector Architecture 1.5 support

WebSphere Application Server V6 supports the J2EE Connector Architecture (JCA) 1.5 specification, which provides new features such as the inbound resource adapter.

From a security perspective, WebSphere Application Server V6 provides an enhanced custom principal and credential programming interface and custom mapping properties at the resource reference level. The custom Java

Authentication and Authorization Service (JAAS) LoginModule, which was developed for JCA principal and credential mapping for WebSphere Application Server V5 or later, continues to be supported.

3.6.4 Secure sockets layer channel framework

The secure sockets layer channel framework incorporates the new IBMJSSE2 implementation and separates the security function of Java Secure Sockets Extension (JSSE) from the network communication function.

3.6.5 Web authentication

WebSphere Application Server V6 enables you to use the JAAS programming model to perform Web authentication in application code. To use this function, you must create your own JAAS login configuration by cloning the WEB_INBOUND login configuration and by defining a cookie=true login option. After a successful login using the login configuration, the Web login session is tracked by single signon (SSO) token cookies. This option replaces the SSOAuthenticator interface, which was deprecated in WebSphere Application Server V4.

3.6.6 Web services security

WebSphere Application Server V6 increases the extensibility of Web services security by providing a pluggable architecture. The implementation in WebSphere Application Server includes many of the features described in the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Version 1 standard. As part of this standard, WebSphere Application Server supports the following:

- Custom, pluggable tokens that are used for signing and encryption.
- Pluggable signing and encryption algorithms.
- Pluggable key locators for locating a key that is used for digital signature or encryption.
- Signing or encrypting elements in a Simple Object Access Protocol (SOAP) message.
- Specifying the order of the signing or encryption processes.

3.7 Deprecated features in this version

The following sections discuss the deprecated functions and features in WebSphere Application Server V6 from the previous version.

3.7.1 Programming model and container support features

Support for the following tsx tags in the JSP engine are deprecated:

- repeat
- dbconnect
- dbquery
- getProperty
- userid
- passwd
- dbmodify

Instead of using these deprecated tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL). JSTL is supported in WebSphere Application Server V6, and the tag library is shipped with the product.

3.7.2 Application services features

The following lists the deprecated application services features and a recommended migration action when available.

- The WebSphere JRAS Extensions API.

  No further enhancements are planned for JRAS support. You should start migrating to the java.util.logging package, especially for any new code that you are writing. For further details about the Java Specification Requests, see:

  `http://www.jcp.org/en/jsr/all`

- The UDDI version 2 EJB interface to the UDDI Registry.

  There is no replacement for the EJB interface. This interface is included in WebSphere Application Server V6 for compatibility with V5 or later. You do not need to take any specific actions and can continue to use the version 2 EJB API. However, you should be aware that WebSphere Application Server does not include any UDDI functionality that is new to UDDI version 3 and that future releases might not support this functionality.

- The UDDI4J version 2 class library, uddi4jv2.jar.

  You should start using the version 3 UDDI APIs. A client library is provided to simplify constructing and sending UDDI v3 requests from Java. This is the

IBM UDDI v3 Client for Java that is provided in uddiv3client.jar. The UDDI4J APIs can still be used, but you should be aware that these APIs do not provide access to any of the new UDDI version 3 functionality and that future release of WebSphere Application Server might not support them.

- All of the low-level UDDI Utility Tools (UUT) APIs, such as BusinessStub, ServiceStub, and so forth.

  These are all replaced by the high-level PromoterAPI interface. You should start using the PromoterAPI in place of these low-level APIs. The PromoterAPI provides the same functionality at a higher level of abstraction.

- The following methods in the JCA runtime:

  – com.ibm.ws.management.descriptor.xml.ConnectionFactory.xml (getPoolContents and getAllPoolContents methods)

  – com.ibm.websphere.j2c.ConnectionManager interface

  – com.ibm.websphere.j2c.ConnectionEventListener interface

  Also, container-managed authentication aliases on a J2C Connection Factory or Datasource are deprecated.

  You should replace:

  – getPoolContents and getAllPoolContents with showPoolContents and whoAllPoolContents.

  – ConnectionManager interface with the JCA 1.5 LazyAssociatableConnectionManager interface.

  – ConnectionEventListener interface the JCA 1.5 LazyEnlistableConnectionManager interface.

  For container-managed authentication aliases, you should specify the container-managed credentials via the application's resource binding information.

- The ApplicationProfile property on the WorkManager panel in the administrative console.

  There is no recommended migration action for this deprecated feature.

- Two items from the DataSource panel in the administrative console:

  – Container-Managed Authentication Alias

  – DefaultPrincipleMapping

  There is no recommended migration action for this deprecated feature.

- All classes in the com.ibm.websphere.servlet.filter package:
  - ChainedRequest
  - ChainedResponse
  - ChainerServlet
  - ServletChain

  You should re-architect you legacy applications to use javax.servlet.filter classes rather than com.ibm.websphere.servlet.filter classes. Starting from the Servlet 2.3 specification, javax.servlet.filter classes give you the capability to intercept requests and examine responses. They also allow you to achieve chaining functionality, as well as embellishing or truncating responses.

- MIME filtering.

  MIME filters were first introduced in WebSphere Application Server V3.5 as a way for servlets to embellish, truncate, or modify the responses that are generated by other servlets, based on the MIME types of the output content.

  The recommended migration actions are as follows:

  - Use javax.servlet.filters, which were introduced in the Servlet 2.3 specification, to plug in filters which can intercept requests to and responses from servlets. These filters also have the capability to modify content flowing in either direction.
  - Use javax.servlet.filters to maintain all the functionality of MIME filters. The javax.servlet.filters are standard APIs and are supported by all compliant application servers.

  For further details about the Java Specification Requests, see:

  `http://www.jcp.org/en/jsr/all`

- Container-managed persistence (CMP) entity beans that are configured with method-level access intent can encounter data access problems such as deadlocks. Therefore, the method level access intent is deprecated.

  You should re-configure CMP entity beans to use bean-level access intent or re-configure application profiles with the Application Server Toolkit.

- All the methods and fields in com.ibm.websphere.product.product and com.ibm.websphere.product.buildInfo classes.

  Thus, the following methods from com.ibm.websphere.product.WASProduct class (which involves com.ibm.websphere.product.product and com.ibm.websphere.product.buildInfo objects) are deprecated:

  - public product getProductByFilename(String basename)
  - public product getProductById(String id)
  - public boolean productPresent(String id)
  - public boolean addProduct(product aProduct)

  - public boolean removeProduct(product aProduct)
  - public Iterator getProducts()
  - public Iterator getProductNames()
  - public String loadVersionInfoAsXMLString(String filename)
  - public String getProductDirName()
  - public static String computeProductDirName()

  You should use the following methods from com.ibm.websphere.product.WASDirectory instead:

  - public WASProductInfo getWASProductInfo(String id)
  - public boolean isThisProductInstalled(String id)
  - public WASProductInfo[] getWASProductInfoInstances()
  - public String getWasLocation()

  Also, instead of getting product information from the old WASProduct API, you should use the following methods in the WASDirectory class to get that information:

  - com.ibm.webpshere.product.WASDirectory.getName(String)
  - com.ibm.webpshere.product.WASDirectory.getVersion(String)
  - com.ibm.webpshere.product.WASDirectory.getBuildLevel(String)
  - com.ibm.webpshere.product.WASDirectory.getBuildDate(String)

- Data Access Beans, which are shipped with WebSphere Application Server in databeans.jar.

  You should use a Service Data Object (SDO) instead of using Data Access Beans.

3.7.3 Security features

The following lists the deprecated security features and a recommended migration action when available.

- SOAP-Security (XML digital signature) that are based on Apache SOAP implementation.

  Instead of using SOAP-Security, you should migrate your application to the JSR-109 implementation of Web service. Also, migrate (reconfigure the application) to use Web Service Security (WSS) 1.0 implementation.

  For further details about the Java Specification Requests, see:

  `http://www.jcp.org/en/jsr/all`

- WSS draft 13 specification-level support is deprecated in favor of the WSS 1.0 implementation.

  You should migrate your applications to the supported WSS 1.0 standard. The draft-level support does not provide interoperability with some third-party vendors, because the message level has been changed between the draft and the WSS 1.0 implementation.

WSS 1.0 is only supported in J2EE 1.4 applications. Thus, you need to migrate applications to J2EE 1.4 first. The next step is to use Application Server Toolkit / Rational Application Developer tooling to reconfigure Web Service Security for the migrated application. There is no automatic migration of Web Service Security in this release of Application Server Toolkit / Rational Application Developer tooling for WebSphere Application Server V6. You have to do the migration manually.

Some Service Provider Interface (SPI) has been deprecated and requires that you migrate to the new SPI for WSS 1.0 support in WebSphere Application Server V6.

Finally, you need to migrate the JAAS LoginModule implementation to the new programming model for the JAAS LoginModule in WebSphere Application Server V6.

3.7.4 System administration features

The following lists the deprecated system administration features and a recommended migration action when available.

- Configuring resources under cell scope.

  You should configure resources under cluster scope. In WebSphere Application Server V6, cell scope resource configuration is discouraged because cell scope resources are visible to every node in the cell, even though not every node in the cell is able to support the resource.

- The depl.extension.reg and installdir options for the install command in the AdminApp scripting object.

  There is no replacement for the depl.extension.reg option. In WebSphere Application Server V5 or later, this option was a non-operational. For the installdir option, use the installed.ear.destination option.

3.7.5 Performance features

The following lists the deprecated performance features and a recommended migration action when available.

- The Performance Monitoring Infrastructure (PMI) Client API, which was introduced in WebSphere Application Server V4.0 to programmatically gather performance data from WebSphere Application Server.
- The Java Management Extension (JMX) interface, which is part of the J2EE specification, is the recommended way to gather WebSphere Application

Server performance data. PMI data can be gathered from the J2EE-managed object MBeans or from the WebSphere PMI Perf MBean. While the J2EE MBeans provide performance data about a specific component, the Perf MBean acts as a gateway to the WebSphere Application Server PMI service, and provides access to the performance data for all the components.

3.8 Application programming model

There are a number of enhancements associated with the application programming model in WebSphere Application Server V6. This section discusses those enhancements.

3.8.1 Programming Model Extensions

Programming Model Extensions (PME) are additional features in the form of services, APIs, development wizards, and deployment extensions. They extend business objects (such as EJBs) by adding functionality. They provide extensions for business processes and workflow functionality for process modelling and implementation tools, such as Process Choreographer. There are also additional next-generation application technologies, such as Extended Messaging, which simplify the development and hosting of applications that need complex business messaging patterns.

The introduction of PMEs began with WebSphere Application Server V4 Enterprise Edition and were consolidated in WebSphere Application Server V5.0 Enterprise Edition. After V5.0, the Enterprise Edition was renamed to WebSphere Business Integration Server Foundation V5.1.

In WebSphere Application Server V6, most of the PMEs are moved into either WebSphere Application Server Express or WebSphere Application Server Network Deployment offerings. Table 3-2 illustrates the Programming Model Extensions distribution across the versions.

Table 3-2 Distribution of PMEs

WebSphere Application Server PMEs	WebSphere Application Server - Express V6	WebSphere Application Server Network Deployment V6	WebSphere Business Integration Foundation Server
Last Participant Support	Yes	Yes	Yes
Internationalization Service	Yes	Yes	Yes
WorkArea Service	Yes	Yes	Yes
ActivitySession Service	Yes	Yes	Yes
Extended JTA support	Yes	Yes	Yes
Startup Beans	Yes	Yes	Yes
Asynchronous Beans (now called WorkManager)	Yes	Yes	Yes
Scheduler Service (now called Time Service)	Yes	Yes	Yes
Object Pools	Yes	Yes	Yes
Dynamic Query	Yes	Yes	Yes
Web Services Gateway Filter Programming Model	No	Yes	Yes
Distributed Map	Yes	Yes	Yes
Application Profiling	Yes	Yes	Yes
CScope Service	Yes	Yes	Yes
Backup Cluster Support	No	Yes	Yes
Dynamic WLM	No	No	Yes
Workflow/Choreographer	No	No	Yes
Business Rule Beans (BRBeans)	No	No	Yes

For more detailed information about Programming Model Extensions, refer to the WebSphere Application Server V6 Information Center at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

3.8.2 J2EE 1.4

WebSphere Application Server V6 incorporates the new J2EE 1.4 specification and maintains compatibility with J2EE 1.2 and 1.3. This section describes the major enhancements associated with the J2EE 1.4 implementation.

Application deployment and packaging

Key changes to J2EE deployment and Packaging in J2EE 1.4 are:

- The use of XML schema instead of DTDs for validating Deployment Descriptors.
- The elimination of the webservicesclient.xml file. References to Web services are included in the web.xml, ejb-jar.xml, and application-client.xml files as service references.

Servlet and JSP changes

J2EE 1.4 includes support for the following:

- Servlet 2.4
- JSP 2.0

Servlet 2.4 offers the following enhancements:

- Request Dispatcher, which now forwards more information about the originating request on to subsequent servlets.
- Filters can be used more selectively and flexibly and related deployment descriptor options are provided.
- Listeners can be registered to react to specific request events.
- There are internationalization enhancements.
- Session time out can now be specified in the deployment descriptor, overriding any application server specific default values.

JSP 2.0 offers the following enhancements:

- A JSP specific expression language, which means that JSPs no longer have to use standard Java expressions. This reduces the need for a JSP programmer to know core Java syntax.
- Functions can be defined by the new expression language, which provides an alternative to custom tags.
- Custom tags are supported, allowing for reuse of bespoke modules of functionality that can be referenced as bespoke tags in the JSP.
- JSTL (JavaServer Pages Standard Tag Libraries) 1.1 is supported because it requires JSP 2.0, specifically the JSP expression language.

Web Services and XML support

J2EE 1.4 now incorporates some of the fundamental Web services Java APIs (and, thus, Web services specifications) as part of the core Enterprise Edition package tree and specification.

The following is a synopsis of the key XML Java Specification Requests from the Java Community Process organization and resulting APIs:

- **JAX-RPC**: A Java API for XML-based remote procedure calls (JSR-101), which allows a program to invoke a remote procedure, function, or method via an XML-based protocol (specifically the SOAP protocol).
- **Web Services for J2EE Architecture** (JSR-109): Defines the programming and runtime architecture for implementing Web services in Java. It mandates the use of JAX-RPC. It also makes reference to other Java API for XML specifications. This Java Specification Request also defines requirements for portability of deployment descriptors which has led in J2EE 1.4 to all deployment descriptors (not just for Web services) to be defined by XML schema documents as well as allowing previous DTDs.
- **JAX-P**: A Java API for XML Parsing (JSRs 5 and 63) which defines how a Java API should be created for parsing XML data.
- **JAX-R**: A Java API for XML Registries (JSR-093). It is a Java API that defines interfaces through which client applications find Web services and also through which Web services and servers publish their interfaces. An example of an XML registry is a UDDI registry.
- **SAAJ**: The SOAP Attachments API for Java (part of JSR-067). This API defines an interfaces for creating and populating SOAP messages that can contain attachments, such as other document formats and mime-types.

The above specifications were implemented in APIs and application server implementations, but they were not previously part of the core J2EE API set of packages or the J2EE specification. J2EE 1.4 directly incorporates them and requires them. The following is a summary of the levels of these items and their supported levels:

- JAXP 1.2
- JAXR 1.0
- JAX-RPC 1.1
- JSR-109, Web Services programming and deployment model
- SAAJ 1.1

For more details about each of these Java Specification Requests, see:

`http://www.jcp.org/en/jsr/all`

Messaging

Pluggable messaging is a revision to the existing specifications and provides additional capabilities. The following is a summary of the latest supported APIs:

- EJB 2.1
  - Typed message beans (used for any inbound JCA including pluggable JMS providers).
  - Timer service Web service end-point support.
- JMS 1.1
  - Unification of point-to-point and pub-sub interfaces.
- J2CA 1.5
  - Inbound connections (supporting pluggable JMS providers, generalized for other types).
  - Resource adapters life cycle support.
  - Work manager (threads for resource adapters).

This section describes in further detail these supported APIs.

EJB 2.1

Extended message-driven beans (MDBs) in EJB 2.1 provide support for message types in addition to JMS messages, called *typed message beans*. An MDB is defined for a single messaging type, in accordance with the message listener interface it employs. Thus, an MDB can have its onMessage method invoked by a JMS message or an email message, for example, depending on the listener interface.

The EJB Timer Service is a container-provided service that allows enterprise beans to be registered for timer callback methods to occur at a specified time, after a specified elapsed time, or after specified intervals.

EJB 2.1 also supports the following new features and functionality:

- Web Services Support. Stateless session beans can have Web service clients according to the J2EE 1.4 specification. The functionality of a stateless session bean can be exposed as a Web service via a WSDL document that the bean implements. EJBs can also invoke Web services.
- The Enterprise Query Language for Entity Beans. Updated in this specification to make the language more similar to SQL and to be more flexible.

JMS

The JMS is a generic Java API that allows an application to access any messaging provider implementation that conforms to the JMS API. JMS 1.1 is part of the J2EE 1.4 specification and API.

Independent software vendor enablement

Also included in Java 1.4 are specifications and APIs that are of concern to independent software vendors (ISVs) who might be constructing application servers and other J2EE environments and tools.

JMX 1.2 / JSR-077 (J2EE Management)

JMX is a specification and API that is used for administrative purposes. For example, behind the WebSphere Application Server V6 administrative console, administrative tasks, and wsadmin command line scripting tools, the JMX API is used to manipulate J2EE entities such as data sources, servers, and so forth. New features include:

- Notification emitters and standard patterns
- Information model representing J2EE application server concepts

Notification emitters are an update and rationalization to the event listener model where objects can register as listeners to notifications of change information coming from management beans.

JSR-088 (J2EE Deployment)

The deployment descriptors in J2EE are now using XML schemas rather than DTDs to validate them, which was influenced by changes in the Web services related specification, XML-based deployment interfaces for J2EE.

For more details about each of these Java Specification Requests:

http://www.jcp.org/en/jsr/detail?id=88

JACC 1.0

As of J2EE 1.4, JACC 1.0 is fully incorporated as part of the specification and package API. This API allows for custom user registries such as LDAP, a local operating system registry, or any registry that can interface with the API to be used as the user authorization registry.

Other J2EE enhancements

The following are enhancements to the Java Database Connectivity and the JavaMail APIs:

- JDBC 3.0, metadata and cursor support
- JavaMail 1.3 updates, mime-type support

3.8.3 Web Services

Web Services have evolved from a convenient solution to a more standard solution as they become more robust and viable. Some of the enhancements in WebSphere Application Server V6 are:

- Portability across J2EE vendor products
- More performance and functional enhancements

WebSphere Application Server V6 adds portability across application server vendors, improves performance as well as functional and QoS enhancements.

Table 3-3 on page 81 shows the differences for Web Services across WebSphere Application Server versions.

Table 3-3 Changes in Web Services across WebSphere Application Server versions

WebSphere Application Server V4 & V5	WebSphere Application Server V5.02 & V5.1	WebSphere Application Server V6
Apache SOAP Used programming model deployment model and engine **Proprietary APIs** No Java standards for Web Services were available **Not WS-I compliant**	**JAX-RPC (JSR-101) 1.0** New standard API for programming Web Services in Java **JSR-109 1.0** New J2EE deployment model for Java Web Services **SAAJ 1.1** **WS-Security** New extensions added **WS-I Basic Profile 1.0** Profile compliance **UDDI4J V2.0 (client)** **Apache SOAP 2.3 enhancements** High performance SOAP engine that supports both HTTP and JMS	**JAX-RPC (JSR-101) 1.1** ▶ Additional type support ▶ xsd:list ▶ Fault support ▶ Name collision rule ▶ New APIs for creating services. ▶ isUserInRole() **JSR-109 - WSEE** ▶ Moved to J2EE 1.4 schema types ▶ Migration of Web Services client DD moving to appropriate containers DDs ▶ Handlers support for EJBs ▶ Service Endpoint Interface (SEI) is a peer to LI/RI **SAAJ 1.2** ▶ APIs for manipulating SOAP XML messages ▶ SAAJ infrastructure now extends DOM (easy to cast to DOM and use) **WS-Security** ▶ OASIS ▶ Follows WS-I Security Profile **WS-TX (WS transactions)** **JAXR support** **UDDI V3 support** ▶ Includes both the registry implementation and the client API library ▶ Client UDDI V3 API different than JAXR (exposes more native UDDI V3 functionality not available in JAXR)

Custom bindings

The following is a summary of enhancements to custom bindings in this release:

- JAX-RPC does not support all XML schema types.
- The custom bindings features allows developers to create their own bindings that are needed to map Java to XML and XML to Java conversions.
- Custom binder code must implement the new CustomBinder interface.
- Custom bindings can be useful where legacy Java types that are unsupported by JAX-RPC must be supported.

Support for generic SOAP elements

In some cases, you might not want mappings to XML schema or custom bindings for special types. You might want to keep the generic SOAPElement type. For example:

- The service can be a conduit to another service like Gateway.
- A handler might need to manipulate the message in a more generic manner.
- Performance can be a priority. In this case, using generic SOAP elements means that you can disable the normal deserialisation processes and avoid its overhead.

Multi-protocol JAX-RPC Support

JAX-RPC clients can invoke Stateless Session EJB as the Web service provider via the RMI-IIOP protocol. This protocol is a more efficient method for calling EJB services.

This enhanced performance is provided without any changes to the JAX-RPC client. Figure 3-1 and Figure 3-2 on page 83 illustrate the existing and the new invocation methods for Stateless Session Beans as Web services.

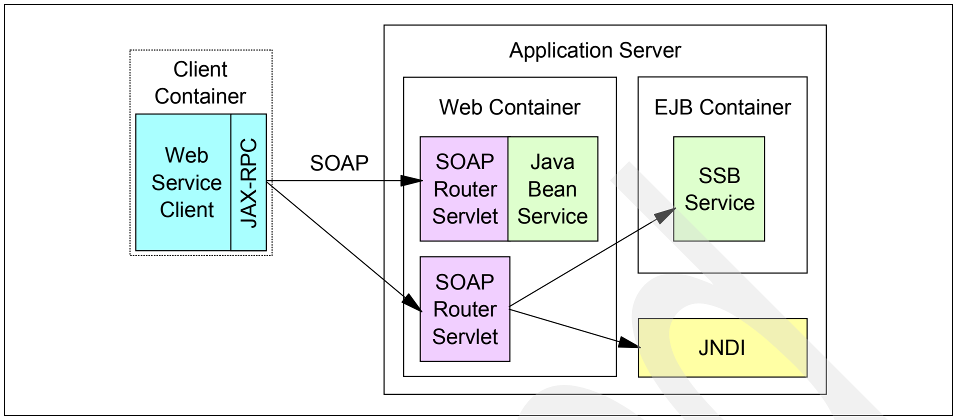


Figure 3-1 Existing SOAP/HTTP invocation

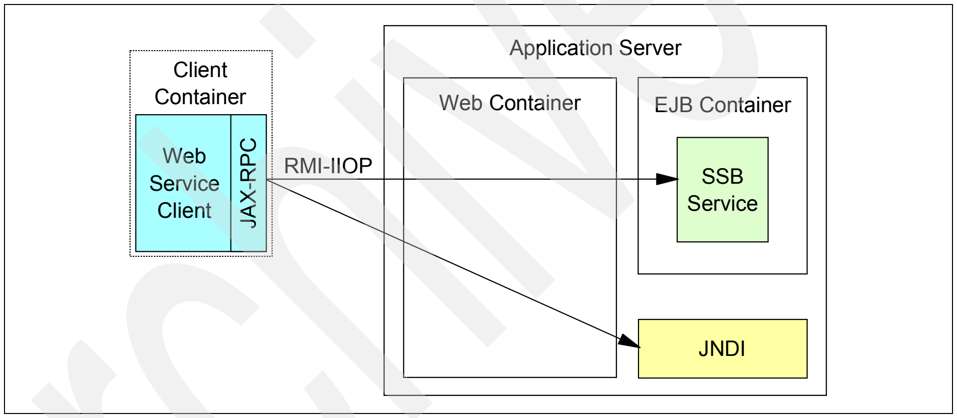


Figure 3-2 New EJB invocation

Client caching

In WebSphere Application Server V5, there was support for server side Web Service caching for Web Services providers running within the application server. In addition to the server side caching, WebSphere Application Server V6 now introduces caching for Web Services client running within an application server including a Web Services Gateway.

As with all caching, this additional caching configuration option can increase performance. Cached data can be used when subsequent identical client calls to the Web service are made. Policies for this caching are highly configurable, and the cache can be invalidated based on configuration rules and time. It can also be invalidated programmatically by the use of the appropriate APIs. Figure 3-3 illustrated the client side Web Services caching.

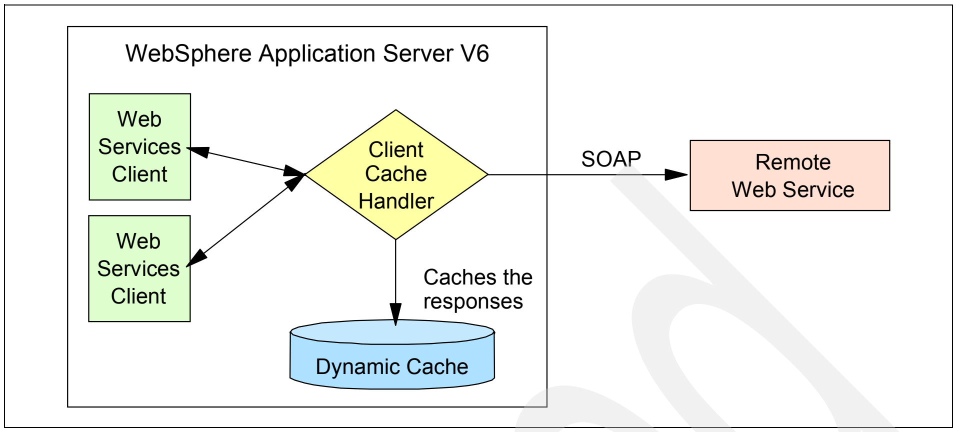


Figure 3-3 Client side Web Services caching

Web Services Gateway enhancements

As a result of the integration of WebSphere Platform Messaging with Web Services, WebSphere Application Server V6 incorporates Web Services Gateway into the application server. (In WebSphere Application Server V5, Web Services Gateway was implemented as a stand-alone application.) This enhancement offers the rich functionality of the WebSphere Platform Messaging for Web Services applications including those used via Web Services Gateway.

Note: Web Services Gateway enhancements are available in WebSphere Application Server Network Deployment V6 package only.

3.8.4 Service Data Object

SDO provides the universal data access model for business data. One of the challenges today is the number of models and APIs that are used for data retrieval and representation (for example, JDBC, XML, JMS, WS, and EIS) and the lack of support for standard application patterns.

To address this issue, SDO provides a unified data representation and retrieval single standard API, across heterogeneous data sources, in a disconnected, source-independent format.

JavaServer Faces (JSF) technology, for example, can use SDO to access data sources rather than concerning the interface developer with the details of the various possible data source access APIs.

3.8.5 JavaServer Faces

JSF technology was introduced in WebSphere Application Server V5.1 and WebSphere Studio Application Developer V5.1.1. At that time, JSF was a beta technology, and JSF .jar files and libraries were packaged with the application.

WebSphere Application Server V6 runtime, Rational Web Developer, and Rational Application Developer tools support JSF V1.0. In addition, JSF .jar files and libraries are included with the runtime environment.

4

WebSphere Application Server architecture

As mentioned previously in Chapter 1, “Introduction to WebSphere Application Server V6” on page 3, WebSphere Application Server V6 implements the J2EE V1.4 specification.

WebSphere Application Server is available in several different configurations that are designed to meet a wide range of customer requirements. Each configuration is packaged with other components that provides a unique environment. At the heart of each configuration is a WebSphere Application Server that provides the runtime environment for enterprise applications.

This chapter covers the following packaging options of WebSphere Application Server:

- WebSphere Application Server - Express V6, referred to as *Express*.
- WebSphere Application Server V6, referred to as WebSphere Application Server.
- WebSphere Application Server Network Deployment V6, referred to as *Network Deployment*.

4.1 Application server configurations

At the heart of each member of the WebSphere Application Server family is an application server with essentially the same architectural structure.

While the application server structure is identical for WebSphere Application Server and Express, there are differences in licensing terms, the development tool provided, and platform support. With WebSphere Application Server and Express, you are limited to stand-alone application servers. Each stand-alone application server provides a fully functional J2EE 1.4 environment.

Network Deployment adds additional elements that allow for more advanced topologies and that provide workload management, scalability, high availability, and central management of multiple application servers.

4.1.1 Stand-alone server configuration

Express, WebSphere Application Server, and Network Deployment all support a single stand-alone server environment. However, with Express and WebSphere Application Server, this is your only option. In a single stand-alone server environment, each application server acts as a unique entity. An application server runs one or more J2EE applications and provides the services that are required to run those applications.

Multiple stand-alone application servers can exist on a machine, either through independent installations of the WebSphere Application Server code or through the creation of multiple configuration profiles within one installation. However, there is no common management or administration provided for multiple application servers. Stand-alone application servers do not provide workload management or failover capabilities.

Figure 4-1 on page 89 shows an architectural overview of a stand-alone application server.

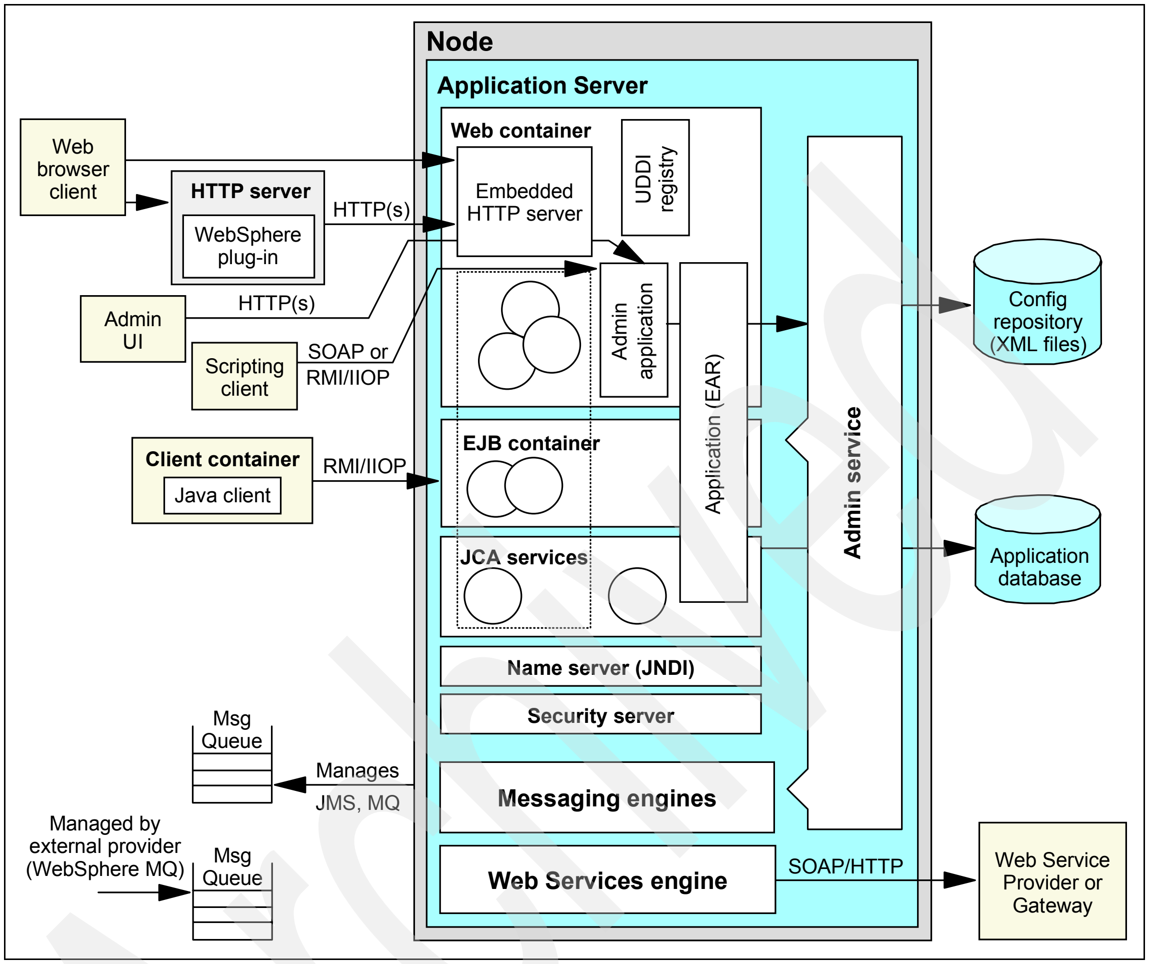


Figure 4-1 Stand-alone server

4.1.2 Distributed server configuration

With Network Deployment, you can build a distributed server configuration, which enables central administration, workload management, and failover. One or more application servers are federated (integrated) into a cell and managed by a deployment manager. The application servers can reside on the same machine as the deployment manager or on multiple separate machines. Administration and management is done centrally from the administration interfaces via the deployment manager.

With this configuration, you can create multiple application servers to run unique sets of applications, and you can manage these servers from a central location. However, more importantly, you can cluster application servers for workload management and failover capabilities. Applications installed to the cluster are replicated across the application servers. When one server fails, another server in the cluster can continue processing. Workload is distributed among Web containers and Enterprise JavaBean (EJB) containers in a cluster using a weighted round-robin scheme.

Figure 4-2 on page 91 illustrates the basic components of an application server in a distributed server environment.

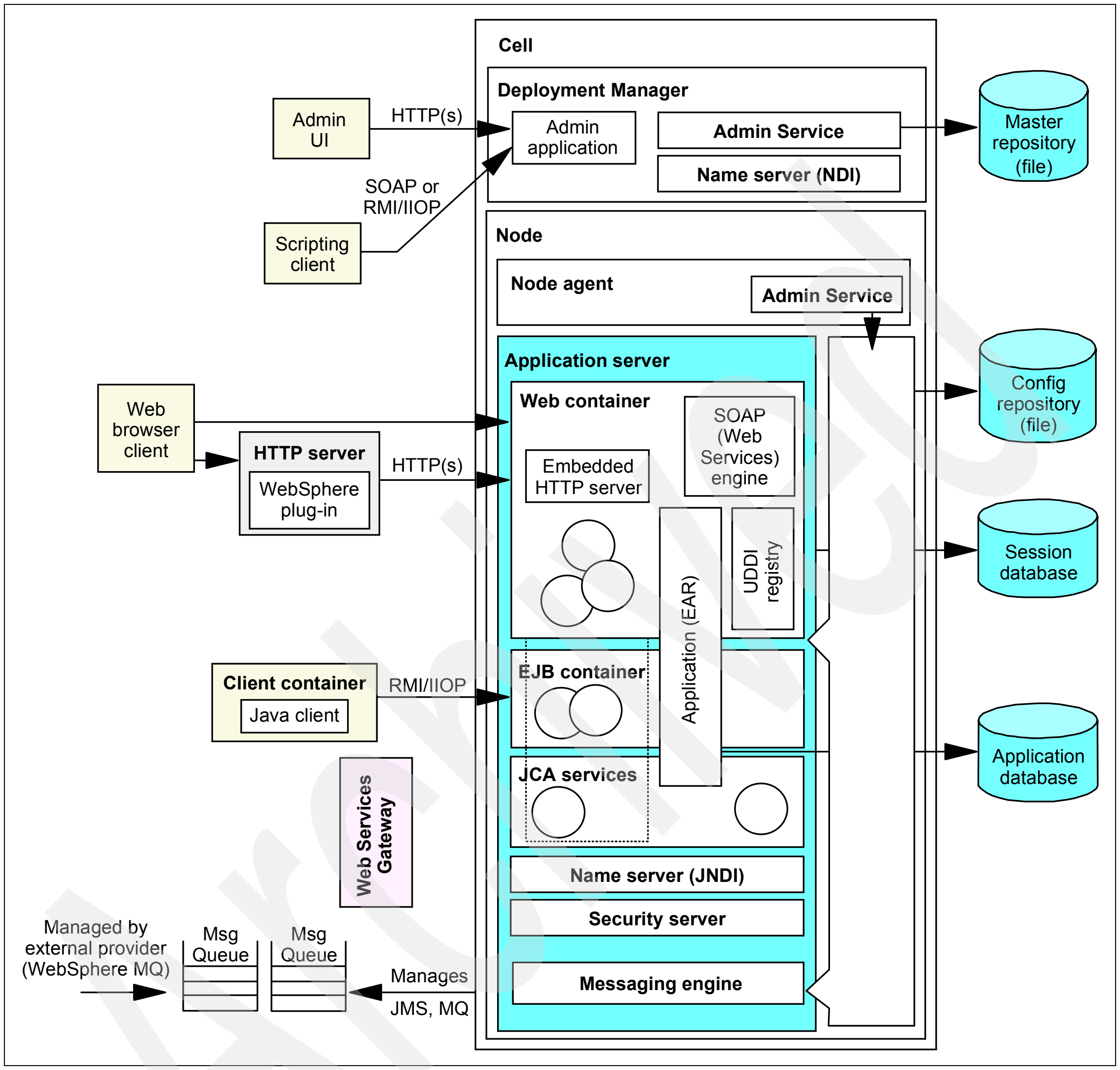


Figure 4-2 Distributed server environment

4.2 Cells, nodes, and servers

Regardless of the configuration, the WebSphere Application Server is organized based on the concept of cells, nodes, and servers. While all these elements are present in each configuration, cells and nodes do not play an important role until you take advantage of the features that are provided with Network Deployment.

Application servers

The application server is the primary runtime component in all configurations and is where the application actually executes. All WebSphere Application Server configurations can have one or more application servers. In the Express and WebSphere Application Server configurations, each application server functions as a separate entity. There is no workload distribution or common administration among application servers. With Network Deployment, you can build a distributed server environment that consists of multiple application servers that are maintained from a central administration point. In a distributed server environment, application servers can be clustered for workload distribution.

Nodes, node groups, and node agents

A *node* is a logical grouping of server processes, managed by WebSphere, that share common configuration and operational control. A node is associated with one physical installation of WebSphere Application Server. In a stand-alone application server configuration, there is only one node.

With Network Deployment, you can configure multiple nodes to be managed from one common administration server. In these centralized management configurations, each node has a *node agent* that works with a deployment manager to manage administration processes.

Node group is a new concept with V6. A node group is a grouping of nodes within a cell that have similar capabilities. It validates that the node is capable of performing certain functions before allowing them. For example, a cluster cannot contain both z/OS and non-z/OS nodes. In this case, multiple node groups would be defined — one for the z/OS nodes and one for the non-z/OS nodes. A DefaultNodeGroup is created automatically and is based on the deployment manager platform. This node group contains the deployment manager and any new nodes with the same platform type.

Cells

A *cell* is a grouping of nodes into a single administrative domain. In the WebSphere Application Server and Express configurations, a cell contains one node. That node can have multiple servers, but the configuration files for each server are stored and maintained individually.

In a distributed server configuration, a cell can consist of multiple nodes, all administered from a single point. The configuration and application files for all nodes in the cell are centralized into a cell master configuration repository. This centralized repository is managed by the deployment manager process and synchronized out to local copies held on each of the nodes.

4.3 Servers

WebSphere Application Server supplies application servers to provide the functions required to host applications. It also provides the ability to define external servers to the administration process. Table 4-1 shows the types of servers that can be defined to the WebSphere Application Server administration tools.

Table 4-1 WebSphere Application Server server support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Application server	Yes	Yes
Application server clustering	No	Yes
External Web server	Yes	Yes
External generic server	No	Yes
WebSphere V5 JMS servers	No	Yes

4.3.1 Application server

Application servers provide the runtime environment for application code. They provide containers and services that specialize in enabling the execution of specific Java application components. Each application server runs in its own Java Virtual Machine (JVM).

4.3.2 Clusters

Network Deployment offers the option of using application server clustering to provide enhanced workload distribution. A *cluster* is a logical collection of application server processes that provides workload balancing and high availability.

Application servers that belong to a cluster are members of that cluster and must have identical application components deployed on them. Other than the applications configured to run on them, cluster members do not have to share any other configuration data.

For example, one cluster member might be running on a large multi-processor server while another member of that same cluster might be running on a small mobile computer. The server configuration settings for each of these two cluster members are very different, except in the area of the application components that are assigned to them. In that area of configuration, they are identical.

The members of a cluster can be located on a single node (vertical cluster), across multiple nodes (horizontal cluster), or on a combination of the two.

When you install, update, or delete an application, the updates are distributed automatically to all members in the cluster. In V5, updating an application on a cluster required stopping the application on every server in the cluster, installing the update, then restarting the server. In V6, you have a rollout update option that updates and restarts the application servers on each node, one node at a time. This capability provides continuous availability of the application.

4.3.3 JMS servers (V5)

In V5, JMS servers provided the default messaging support for WebSphere Application Server. For migration purposes, Network Deployment in V6 supports cells that contain both V5 and V6 nodes (the deployment manager must be at V6) and, by extension, supports existing JMS servers in V5 application servers in the cell.

4.3.4 External servers

You can define servers other than WebSphere application servers to the administrative process. The servers that you can define are:

- Generic servers

  A generic server is a server that is managed in the WebSphere administrative domain, although it is not a server that is supplied by WebSphere Application Server. The generic server can be any server or process that is necessary to support the application server environment, including a Java server, a C or C++ server or process, a CORBA server, or a Remote Method Invocation (RMI) server.

- Web servers

  Web servers can be defined to the administration process as a Web server node, allowing you to associate applications with one or more defined Web servers. Web server nodes can be *managed* or *unmanaged*.

  Managed nodes have a node agent on the Web server machine, allowing the deployment manager to administer the Web server. The Web server can be started or stopped from the deployment manager. The Web server plug-in for the node can be generated and pushed automatically to the Web server.

  Unmanaged Web server nodes are not managed by WebSphere. You would normally find these outside the firewall or in the DMZ. Web server plug-in files must be copied manually or FTP’d to the Web server. However, defining the Web server as a node allows you to generate custom plug-in files for it.

As a special case, if the unmanaged Web server is an IBM HTTP Server, you can administer the Web server from the WebSphere administrative console and push the plug-in configuration file automatically to the Web server. This is done by the deployment manager using HTTP commands to the IBM HTTP Server administration process and does not require a node agent.

4.4 Containers

The J2EE 1.4 specification defines the concept of containers to provide runtime support for applications. There are two containers in the application server implementation:

- Web container, which processes HTTP requests, servlets and JSPs
- EJB container, which processes enterprise beans (EJBs)

In addition, there is an application client container that can run on the client machine. Table 4-2 shows the containers that WebSphere Application Server supports.

Table 4-2 WebSphere Application Server container support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Web container	Yes	Yes
EJB container	Yes	Yes
Application client container	Yes	Yes

4.4.1 Web container

The Web container processes servlets, JSP files and other types of server-side includes. Each application server runtime has one logical Web container, which can be modified but not created or removed. Each Web container provides the following:

- Web container transport chains

  Requests are directed to the Web container using the Web container inbound transport chain. The chain consists of a TCP inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP

1.0 and 1.1 requests, and a Web container channel over which requests for servlets and JSPs are sent to the Web container for processing.

- Servlet processing

  When handling servlets, the Web container creates a request object and a response object, then invokes the servlet service method. The Web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

- HTML and other static content processing

  Requests can be sent directly to the Web container by directing them to the Web container inbound transport chain. In this case, HTML and static content will be served Webcontainer inbound chain. This method is useful for testing or development purposes. In the simplest of configurations, this method can even be appropriate for production use. However, in most cases, the use of an external Web server and Web server plug-in as a front-end to the Web container is more appropriate for a production environment.

- Session management

  Support is provided for the javax.servlet.http.HttpSession interface described in the Servlet API specification.

- Web services engine

  Web services are provided as a set of APIs in cooperation with the J2EE applications. Web services engines are provided to support SOAP.

Web server plug-ins

Although the Web container can serve static content, a more likely scenario is that an external Web server will be used to receive client requests. The Web server can serve requests that do not require any dynamic content, for example, HTML pages. However, when a request requires dynamic content (JSP/servlet processing), it must be forwarded to WebSphere Application Server for handling.

The mechanism to accomplish this is provided in the form of a Web server plug-in. The plug-in is included with the WebSphere Application Server packages for installation on a Web server. An XML configuration file, configured on the WebSphere Application Server, is copied to the Web server plug-in directory. The plug-in uses the configuration file to determine whether a request should be handled by the Web server or an application server. When a request for an application server is received, it is forwarded to the appropriate Web container in the application server. The plug-in can use HTTP or HTTPs to transmit the request.

4.4.2 EJB container

The EJB container provides the runtime services that are needed to deploy and manage enterprise beans. It is a server process that handles requests for both session and entity beans.

The enterprise beans (packaged in EJB modules) that are installed in an application server do not communicate directly with the server. Instead, the EJB container provides an interface between the EJBs and the server. Together, the container and the server provide the bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative viewpoint, the container manages data storage and retrieval for the contained beans. A single container can host more than one EJB JAR file.

4.4.3 Client application container

The client application container is a separately installed component on the client's machine. It allows the client to run applications in an EJB-compatible J2EE environment. You use a command-line executable (`launchClient`) to launch the client application along with its client container runtime.

4.5 Application server services

The application server provides other services in addition to the containers. Table 4-3 lists these other services.

Table 4-3 WebSphere Application Server services

	WebSphere Application Server - Express and WebSphere Application Server	WebSphere Application Server Network Deployment
JCA services	Yes	Yes
Transaction service	Yes	Yes
Dynamic cache service	Yes	Yes
Message listener service	Yes	Yes
ORB service	Yes	Yes
Administration service (JMX)	Yes	Yes

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Diagnostic trace service	Yes	Yes
Debugging service	Yes	Yes
Name service (JNDI)	Yes	Yes
Performance Monitoring Infrastructure (PMI) service	Yes	Yes
Security service (JAAS and Java 2 security)	Yes	Yes
Service integration bus	Yes	Yes
Application profiling service[1]	Yes	Yes
Compensation service[1]	Yes	Yes
Internationalization service[1]	Yes	Yes
Object pool service[1]	Yes	Yes
Startup beans service[1]	Yes	Yes
Activity session service[1]	Yes	Yes
Work area partition service[1]	Yes	Yes
Work area service[1]	Yes	Yes

1. These services support the PMEs that are outlined in Table 4-12 on page 150.

4.5.1 JCA services

Connection management for access to enterprise information systems (EIS) in WebSphere Application Server is based on the J2EE Connector Architecture (JCA) specification. The connection between the enterprise application and the EIS is done through the use of EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts that exist between the application server and the EIS.

The Connection Manager in the application server pools and manages connections. It is capable of managing connections obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC 2.0 Extensions (and later) specification.

4.5.2 Transaction service

WebSphere applications can use transactions to coordinate multiple updates to resources as one unit of work such that all or none of the updates are made permanent. Transactions are started and ended by applications or the container in which the applications are deployed.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service protocol (for example, application servers) or Web Service Atomic Transaction protocol.

WebSphere Application Server also participates in transactions that are imported through J2EE Connector 1.5 resource adapters. WebSphere applications can be configured to interact with (or to direct the WebSphere transaction service to interact with) databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction coordination is not required.

The way that applications use transactions depends on the type of application component, as follows:

- A session bean can either use container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) use bean-managed transactions.

In WebSphere Application Server, transactions are handled by three main components:

- A transaction manager that supports the enlistment of recoverable XAResources and ensures that each such resource is driven to a consistent outcome either at the end of a transaction or after a failure and restart of the application server.
- A container in which the J2EE application runs. The container manages the enlistment of XAResources on behalf of the application when the application performs updates to transactional resource managers (for example, databases). Optionally, the container can control the demarcation of transactions for enterprise beans configured for container-managed transactions.

- An application programming interface (UserTransaction) that is available to bean-managed enterprise beans and servlets, allowing such application components to control the demarcation of their own transactions.

4.5.3 Dynamic cache service

The dynamic cache service improves performance by caching the output of servlets, commands, Web services, and JSP files. The dynamic cache works within an application server, intercepting calls to cachable objects, for example, through a servlet's service() method or a command's execute() method. The dynamic cache either stores the object's output to or serves the object's content from the dynamic cache.

Because J2EE applications have high read-write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability.

The following caching features are available in WebSphere Application Server.

- Cache replication

  Cache replication among cluster members takes place using the WebSphere data replication service. Data is generated one time and copied or replicated to other servers in the cluster, thus saving execution time and resources.

- Cache disk offload

  By default, when the number of cache entries reaches the configured limit for a given WebSphere server, eviction of cache entries occurs which allows new entries to enter the cache service. The dynamic cache includes an alternative feature named disk offload, which copies the evicted cache entries to disk for potential future access.

- Edge Side Include caching

  The Web server plug-in contains a built-in ESI processor. The ESI processor has the ability to cache whole pages, as well as fragments, providing a higher cache hit ratio. The cache implemented by the ESI processor is an in-memory cache, not a disk cache. Therefore, the cache entries are not saved when the Web server is restarted.

- External caching

  The dynamic cache has the ability to control caches outside of the application server, such as that provided by the Edge components, a non-z/OS IBM HTTP Server's FRCA cache and a non-z/OS WebSphere HTTP Server plug-in ESI Fragment Processor. When external cache groups are defined, the dynamic cache matches externally cacheable cache entries with those groups and pushes cache entries and invalidations out to those groups. This allows WebSphere to manage dynamic content beyond the application server. The content can then be served from the external cache, instead of the application server, improving savings in performance.

4.5.4 Message listener service

With EJB 2.1, an ActivationSpec is used to connect message-driven beans (MDBs) to destinations. However, existing EJB 2.0 MDBs can be deployed against a listener port as in WebSphere V5. For those MDBs, the message listener service provides a listener manager that controls and monitors one or more JMS listeners. Each listener monitors a JMS destination on behalf of a deployed MDB.

4.5.5 Object Request Broker service

An Object Request Broker (ORB) manages the interaction between clients and servers, using IIOP. It enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client, providing location transparency. The client calls an operation on a local object, known as a stub. Then the stub forwards the request to the desired remote object, where the operation is run and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and any required parameter and for sending the request in the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub. The stub, in turn, returns to the client application, as though the operation had been run locally.

WebSphere Application Server uses an ORB to manage communication between client applications and server applications as well as communication among product components.

4.5.6 Administrative service

The administrative service runs within each server JVM. In WebSphere Application Server and Express, the administrative service runs in the application server. In the Network Deployment configuration, each of the following hosts an administrative service:

- Deployment manager
- Node agent
- Application server

The administrative service provides the necessary functions to manipulate configuration data for the server and its components. The configuration is stored in a repository in the server's file system.

The administrative service has a security control and filtering function that provides different levels of administration to certain users or groups using the following administrative roles:

- Administrator
- Monitor
- Configurator
- Operator

4.5.7 Name service

Each application server hosts a name service that provides a Java Naming and Directory Interface (JNDI) name space. The service is used to register resources that the application server hosts. The JNDI implementation in WebSphere Application Server is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

JNDI provides the client-side access to naming and presents the programming model that is used by application developers. CosNaming provides the server-side implementation and is where the name space is actually stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming and interacts with the CosNaming server on behalf of the client.

The naming architecture is used by clients of WebSphere applications to obtain references to objects related to those applications. These objects are bound into a mostly hierarchical structure, referred to as a *name space*. The name space structure consists of a set of name bindings, each consisting of a name relative to a specific context and the object bound with that name. The name space can be accessed and manipulated through a name server.

The following are features of a WebSphere Application Server name space:

- The name space is distributed

  For additional scalability, the name space for a cell is distributed among the various servers. The deployment manager, node agent, and application server processes all host a name server.

The default initial context for a server is its server root. System artifacts, such as EJB homes and resources, are bound to the server root of the server with which they are associated.

- Transient and persistent partitions

  The name space is partitioned into transient areas and persistent areas. Server roots are transient. System-bound artifacts such as EJB homes and resources are bound under server roots. There is a cell persistent root, which can be used for cell-scoped persistent bindings, and a node persistent root, which can be used to bind objects with a node scope.

- Federated name space structure

  A name space is a collection of all names bound to a particular name server. A name space can contain naming context bindings to contexts located in other servers. If this is the case, the name space is said to be a *federated name space*, because it is a collection of name spaces from multiple servers. The name spaces link together to form cooperatively a single logical name space.

  In a federated name space, the real location of each context is transparent to client applications. Clients have no knowledge that multiple name servers are handling resolution requests for a particular requested object.

  In the Network Deployment distributed server configuration, the name space for the cell is federated among the deployment manager, node agents, and application servers of the cell. Each such server hosts a name server. All name servers provide the same logical view of the cell name space, with the various server roots and persistent partitions of the name space being interconnected by means of the single logical name space.

- Configured bindings

  The configuration graphical interface and script interfaces can be used to configure bindings in various root contexts within the name space. These bindings are read-only and are bound by the system at server startup.

- Support for CORBA Interoperable Naming Service object URLs

  WebSphere Application Server contains support for CORBA object URLs (corbaloc and corbaname) as JNDI provider URLs and lookup names.

Figure 4-3 on page 104 summarizes the naming architecture and its components.

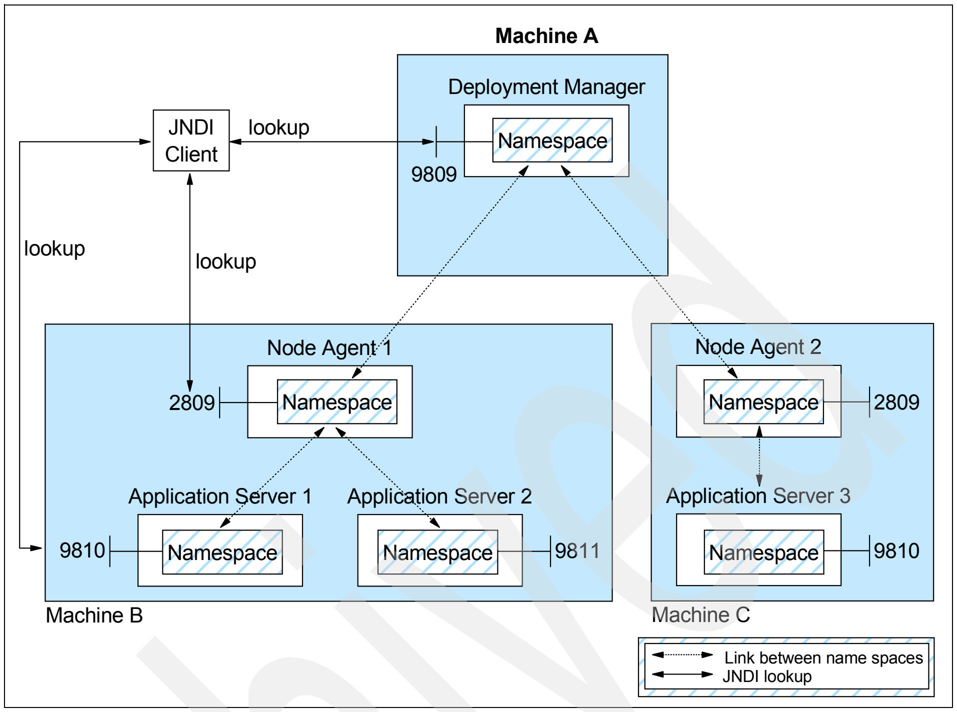


Figure 4-3 Naming topology

4.5.8 PMI service

WebSphere Application Server collects data on runtime and applications through the Performance Monitoring Infrastructure (PMI). This infrastructure is compatible with and extends the JSR-077 specification.

PMI uses a client-server architecture. The server collects performance data from various WebSphere Application Server components and stores it in memory. This data consists of counters such as servlet response time and data connection pool usage. The data can then be retrieved using a Web client, Java client or JMX client. WebSphere Application Server contains Tivoli Performance Viewer, now integrated in the WebSphere administrative console, which displays and monitors performance data.

WebSphere Application Server also collects data by timing requests as they travel through the product components. PMI request metrics log time spent in major components, such as Web containers, EJB containers, and databases. These data points are recorded in logs and can be written to Application Response Time (ARM) agents used by Tivoli monitoring tools.

4.5.9 Security service

Each application server JVM hosts a security service that uses the security settings held in the configuration repository to provide authentication and authorization functionality.

4.6 Data Replication Service

The Data Replication Service (DRS) is responsible for replicating in-memory data among WebSphere processes.

You can use DRS for:

- HTTP session persistence and failover.
- Stateful session EJB persistence and failover (new in V6)
- Dynamic cache replication

Replication is done through the use of replication domains consisting of server or cluster members that have a need to share internal data. Multiple domains can be used, each for a specific task among a set of servers or clusters. While HTTP session replication and EJB state replication can (and should) share a domain, a separate domain is needed for dynamic cache replication.

A domain can be defined so that each domain member has a single replicator that sends data to another member of the domain or each member has multiple replicators that send data to multiple members of the domain.

Two topology options are offered:

- Peer-to-peer topology

  Each application server stores sessions in its own memory. It also stores sessions to and retrieves sessions from other application servers. In other words, each application server acts as a client by retrieving sessions from other application servers, and each application server acts as a server by providing sessions to other application servers. This mode, working in conjunction with the workload manager, provides hot failover capabilities.

- Client/Server topology

  Application servers act as either a replication client or a server. Those that act as replication servers store sessions in their own memory and provide session information to clients. They are dedicated replication servers that just store sessions but do not respond to the users' requests. Client application servers send session information to the replication servers and retrieve sessions from the servers. They respond to user requests and store only the sessions of the users with whom they interact.

4.7 Virtual hosts

A virtual host is a configuration enabling a single host machine to resemble multiple host machines. It allows a single physical machine to support several independently configured and administered applications. It is not associated with a particular node. It is a configuration, rather than a “live object,” which is why it can be created but not started or stopped.

Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP host name and port number that is used to request the servlet, for example yourHostName:80. When a servlet request is made, the server name and port number entered into the browser are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the servlet. If no match is found, an HTTP 404 error is returned to the browser.

WebSphere Application Server provides two default virtual hosts:

- default_host

  Used for accessing most applications. The default settings for default_host map to all requests for any alias on ports 80, 9443, and 9080.

  For example:

  ```
  http://localhost:80/snoop
  http://localhost:9080/snoop
  ```

- admin_host

  Specifically configured for accessing the WebSphere Application Server administrative console. Other applications are not accessible through this virtual host. The default settings for admin_host map to requests on ports 9060 and 9043.

  For example:

  ```
  http://localhost:9060/admin
  ```

4.8 Session management

In many Web applications, users dynamically collect data as they move through the site based on a series of selections on pages they visit. Where the user goes next and what the application displays as the user's next page (or next choice) might depend on what the user has chosen previously from the site. In order to maintain this data, the application stores it in a *session*.

WebSphere supports three approaches to track sessions:

- SSL session identifiers, where SSL session information is used to track the HTTP session ID.
- Cookies, where the application server session support generates a unique session ID for each user and returns this ID to the user's browser using a cookie. The default name for the session management cookie is JSESSIONID. Using cookies is the most common method of session management.
- URL rewriting

Session data can be kept in local memory cache, stored externally on a database, or kept in memory and replicated among application servers.

Table 4-4 shows the session support for each WebSphere Application Server configuration.

Table 4-4 WebSphere Application Server session management support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Cookies	Yes	Yes
URL rewriting	Yes	Yes
SSL session identifiers	Yes	Yes
In memory cache	Yes	Yes
Session persistence using a database	Yes	Yes
Memory-to-memory session persistence	No	Yes

The Servlet 2.4 specification defines the session scope at the Web application level, meaning that session information can only be accessed by a single Web application. However, there can be times when there is a logical reason for multiple Web applications to share information, for example a user name. WebSphere Application Server provides an IBM extension to the specification, allowing session information to be shared among Web applications within an enterprise application. This option is offered as an extension to the application deployment descriptor, and no code change is necessary to enable this option. You specify this option during application assembling.

4.8.1 HTTP Session persistence

Many Web applications use the simplest form of session management, the in-memory local session cache. The local session cache keeps session information in memory and local to the machine and WebSphere Application Server where the session information was first created. Local session management does not share user session information with other clustered machines. Users only obtain their session information if they return to the machine and WebSphere Application Server holds their session information about subsequent accesses to the Web site.

Most importantly, local session management lacks a persistent store for the sessions it manages. A server failure takes down not only the WebSphere instances running on the server but also destroys any sessions managed by those instances.

By default, WebSphere Application Server places session objects in memory. However, the administrator has the option of enabling persistent session management. This instructs WebSphere to place session objects in a persistent store. Using a persistent store allows the user session data to be recovered by an application server on restart or another cluster member after a cluster member in a cluster fails or is shut down. Two options for HTTP session persistence are available:

- Database

  Session information can be stored in a central session database for session persistence.

  In a single-server environment, the session can be persisted when the user's session data must be maintained across a server restart or when the user's session data is too valuable to lose through an unexpected server failure.

  In a multi-server environment, the multiple application servers hosting a particular application need to share this database information in order to maintain session states for the stateful components.

- Memory-to-memory using data replication services

  In a Network Deployment distributed server environment, WebSphere internal replication enables sessions to be shared among application servers without using a database. Using this method, sessions are stored in the memory of an application server, providing the same functionality as a database for session persistence.

4.8.2 Stateful session EJB persistence

V6 offers failover capability of stateful session EJBs. This function uses data replication services and interacts with the workload manager component during a failover situation.

4.9 Web services

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. WebSphere Application Server can act as both a Web service provider and as a requester. As a provider, it hosts Web services that are published for use by clients. As a requester, it hosts applications that invoke Web services from other locations.

WebSphere Application Server supports SOAP-based Web service hosting and invocation.

Table 4-5 WebSphere Application Server server support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Web services support	Yes	Yes
Private UDDI v3 Registry	Yes	Yes
Web Services Gateway	No	Yes
Enterprise Web services	Yes	Yes

The Web services support includes the following:

- Web Services Description Language (WSDL), an XML-based description language that provides a way to catalog and describe services. It describes the interface of Web services (parameters and result), the binding (SOAP, EJB), and the implementation location.
- Universal Discovery Description and Integration (UDDI), a global, platform-independent, open framework to enable businesses to discover each other, define their interaction, and share information in a global registry.

  UDDI support in WebSphere Application Server V6 includes UDDI V3 APIs, some UDDI V1 and V2 APIs, UDDI V3 client for Java, and UDDI4J for compatibility with UDDI V2 registries. It also provides a UDDI V3 Registry, that is integrated in WebSphere Application Server.

- Simple Object Access Protocol (SOAP), a lightweight protocol for exchange of information in a decentralized, distributed environment.
- Extensible Markup Language (XML), which provides a common language for exchanging information.
- JAX-RPC (JSR-101), which provides the core programming model and bindings for developing and deploying Web services on the Java platform. It is a Java API for XML-based RPC and supports JavaBeans and enterprise beans as Web service providers.
- Enterprise Web services (JSR-109), which adds EJBs and XML deployment descriptors to JSR-101.
- WS-Security, the specification that covers a standard set of SOAP extensions that can be used when building secure Web services to provide integrity and confidentiality. It is designed to be open to other security models, including PKI, Kerberos, and SSL. WS-Security provides support for multiple security tokens, multiple signature formats, multiple trust domains, and multiple encryption technologies. It includes security token propagation, message integrity, and message confidentiality. The specification is proposed by IBM, Microsoft, and VeriSign for review and evaluation. In the future, it will replace existing Web services security specifications from IBM and Microsoft including SOAP Security Extensions (SOAP-SEC), Microsoft's WS-Security and WS-License, as well as security token and encryption documents from IBM.
- JAXR, an API that standardizes access to Web services registries from within Java. The current JAXR version, 1.0, defines access to ebXML and UDDI V2 registries. WebSphere Application Server provides JAXR level 0 support, meaning it supports UDDI registries.

  JAXR does not map precisely to UDDI. For a precise API mapping to UDDI V2, IBM provides UDDI4J and IBM Java Client for UDDI v3.
- SAAJ, the SOAP with Attachments API for Java (SAAJ) defines a standard for sending XML documents over the Internet from the Java platform.

IBM value add: In addition to the requirements of the specifications, IBM has added the following value add features to its Web services support:

- JAX-RPC does not support all XML schema types. Custom bindings allow developers to create custom bindings that are needed to map Java to XML and XML to Java conversions.
- In cases where you might not want mapping to XML schema or custom binding but want to keep it generic, support for generic SOAP elements allows you to eliminate binding and instead use the generic SOAPElement type.
- Multi-protocol support for a stateless session EJB as the Web service provider providing enhanced performance without any changes to JAX-RPC client.
- In WebSphere Application Server V5, there was support for server-side Web service caching for Web services providers running within the application server. In addition to this server side caching, V6 introduces caching for Web services clients running within a V6 application server, including the Web Services Gateway.

4.9.1 Enterprise services (JCA Web services)

Enterprise services offer access over the Internet to applications in a platform-neutral and language-neutral fashion. They offer access to enterprise information systems (EIS) and message queues and can be used in a client/server configuration without the Internet. Enterprise services can access applications and data on a variety of platforms and in a variety of formats.

An enterprise service wraps a software component in a common services interface. The software component is typically a Java class, EJB, or JCA resource adapter for an EIS. In services terminology, this software component is known as the implementation. Enterprise services primarily use WSDL and WSIF to expose an implementation as a service.

Using the Integrated Edition of WebSphere Studio, you can turn CICS and IMS transactions using J2EE Connector Architecture (JCA) into Web services.

4.9.2 Web service client

Applications that invoke Web services are known as Web service clients or Web service requestors. An application that acts as a Web service client is deployed to WebSphere Application Server like any other enterprise application. No additional configuration or software is needed for the Web services client to function. Web services clients can also be stand-alone applications.

A Web service client will bind to a Web service server to invoke the Web service. The binding is done using a service proxy (or *stub*), which is generated based on the WSDL document for the Web service. This service proxy contains all the needed information to invoke the Web service and is used locally by the clients to access the business service. The binding can also be done dynamically using WSIF.

4.9.3 Web service provider

An application that acts as a Web service is deployed to WebSphere Application Server like any other enterprise application. The Web services are contained in Web modules or EJB modules.

Publishing the Web service to a UDDI registry makes it available to anyone searching for it. Web services can be published to a UDDI registry using the Web Services Explorer provided with Rational Application Developer.

When using Rational Application Developer to package the application for deployment, no additional configuration or software is needed for the Web services client to function. The SOAP servlets are added automatically and a SOAP administrative tool is included in a Web module.

If not, you can use endptEnabler command-line tool found in the WebSphere bin directory to enable the SOAP services within the EAR file and add the SOAP administrative tool.

4.9.4 Enterprise Web Services

The Enterprise Web Services, based on the JSR-109 specification request, provides the use of JAX-RPC in a J2EE environment defining the runtime architecture, as well as the implementation and deployment of Web services in a generic J2EE server. The specification defines the programming model and architecture for implementing Web services in Java based on JSRs 67, 93, 101, and future Java Specification Requests (JSRs) that are related to Web services standards. You can find a list of JSRs at:

http://www.jcp.org/en/jsr/all

4.9.5 IBM WebSphere UDDI Registry

WebSphere Application Server V6 provides a private UDDI registry that implements V3.0 of the UDDI specification. This enables the enterprise to run its own Web services broker within the company or provide brokering services to the outside world. The UDDI registry installation and management is now integrated in with WebSphere Application Server.

There are three interfaces to the registry for inquiry and publish:

- Through the UDDI SOAP API.
- Through the UDDI EJB client interface.
- Through the UDDI user console. This Web-based GUI interface can be used to publish and to inquire about UDDI entities but only provides a subset of the UDDI API functions.

Security for the UDDI registry is handled using WebSphere security. To support the use of secure access with the IBM WebSphere UDDI Registry, you need to configure WebSphere to use HTTPS and SSL.

A relational database is used to store registry data.

4.9.6 Web Services Gateway

The Web Services Gateway bridges the gap between Internet and intranet environments during Web service invocations. The gateway builds upon the Web services Definition Language (WSDL) and the Web Services Invocation Framework (WSIF) for deployment and invocation.

With V6, the Web Services Gateway is fully integrated into the integration service technologies which provides the runtime. The administration is done directly from the WebSphere administrative console.

The primary function of the Web Services Gateway is to map an existing WSDL-defined Web service (target service) to a new service (gateway service) that is offered by the gateway to others. The gateway thus acts as a proxy. Each target service, whether internal or external is available at a service integration bus destination.

The role formerly played by filters in the V5 Web Services Gateway is now provided by through JAX-RPC handlers. The use of JAX-RPC handlers provides a standard approach for intercepting and filtering service messages. JAX-RPC handlers interact with messages as they pass into and out from the service integration bus. Handlers monitor messages at ports, and take appropriate action depending upon the sender and content of each message.

Exposing internal Web services to the outside world

Web services hosted internally and made available through the service integration bus are called *inbound services*. Inbound services are associated with a service destination. Service requests and responses are passed to the service through an endpoint listener and associated inbound port.

From the gateway's point of view, the inbound service is the target service. To expose the target service for outside consumption, the gateway takes the WSDL file for the inbound service and generates a new WSDL file that can be shared with outside requestors. The interface described in the WSDL is exactly the same, but the service endpoint is changed to the gateway, which is now the official endpoint for the service client.

Figure 4-4 on page 114 diagrams how a Web service is exposed to the outside world.

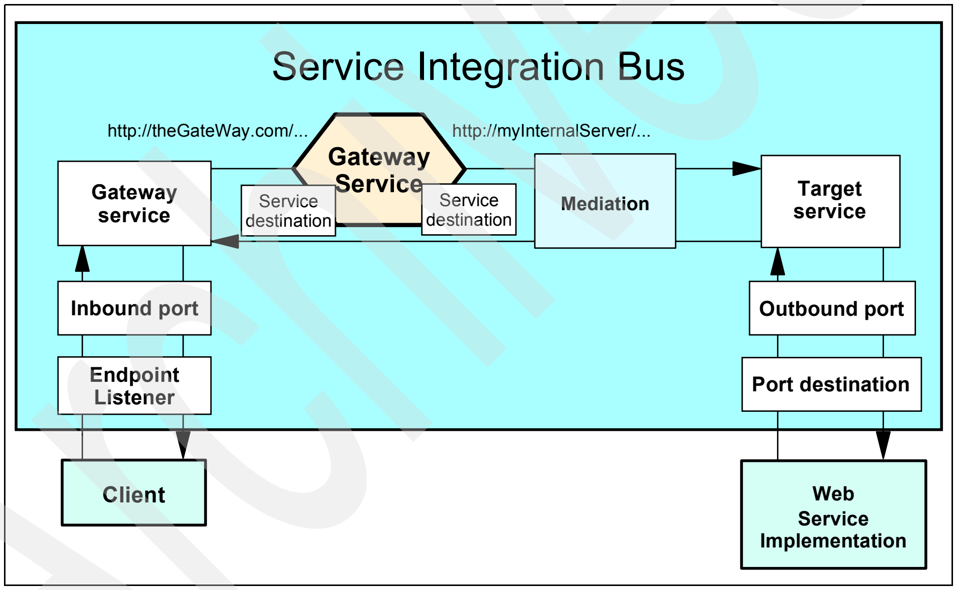


Figure 4-4 Exposing Web services through the Gateway

Externally hosted Web services

A Web service that is hosted externally and made available through the service integration bus is called an *outbound service*. To configure an externally-hosted service for a bus, you first associate it with a service destination, then you configure one or more port destinations (one for each type of binding, for example SOAP over HTTP or SOAP over JMS) through which service requests and responses are passed to the external service.

From the gateway’s point of view, the outbound service is the target service. Mapping a gateway service to the target service will allow internal service requestors invoke the service as though it were running on the gateway. Again, a new WSDL is generated by the gateway showing the same interface but naming the gateway as service provider rather than the real internal server. All requests to the gateway service are rerouted to the actual implementation specified in the original WSDL.

Of course, every client could access external Web services by traditional means, but if you add the gateway as an additional layer in between, clients do not have to change anything if the service implementor changes. This scenario is very similar to the illustration in Figure 4-4, with the difference that the Web service implementation is located at a site on the Internet.

UDDI publication and lookup

The gateway facilitates working with UDDI registries. As you map a service for external consumption using the gateway, you can publish the exported WSDL in the UDDI registry. When the services in the gateway are modified, the UDDI registry is updated with the latest changes.

4.10 Service integration bus

The service integration bus provides the communication infrastructure for messaging and service-oriented applications, thus unifying this support into a common component. The service integration bus provides a JMS 1.1 compliant JMS provider for reliable message transport and has the capability of intermediary logic to intelligently adapt message flow in the network. It also supports the attachment of Web services requestors and providers.

The service integration bus capabilities have been fully integrated within WebSphere Application Server, enabling it to take advantage of WebSphere security, administration, performance monitoring, trace capabilities, and problem determination tools.

The service integration bus is often referred to as just a *bus*. When used to host JMS applications, it is also often referred to as a *messaging bus*.

Figure 4-5 on page 116 illustrates the service integration bus and how it fits into the larger picture of an Enterprise Service Bus (ESB).

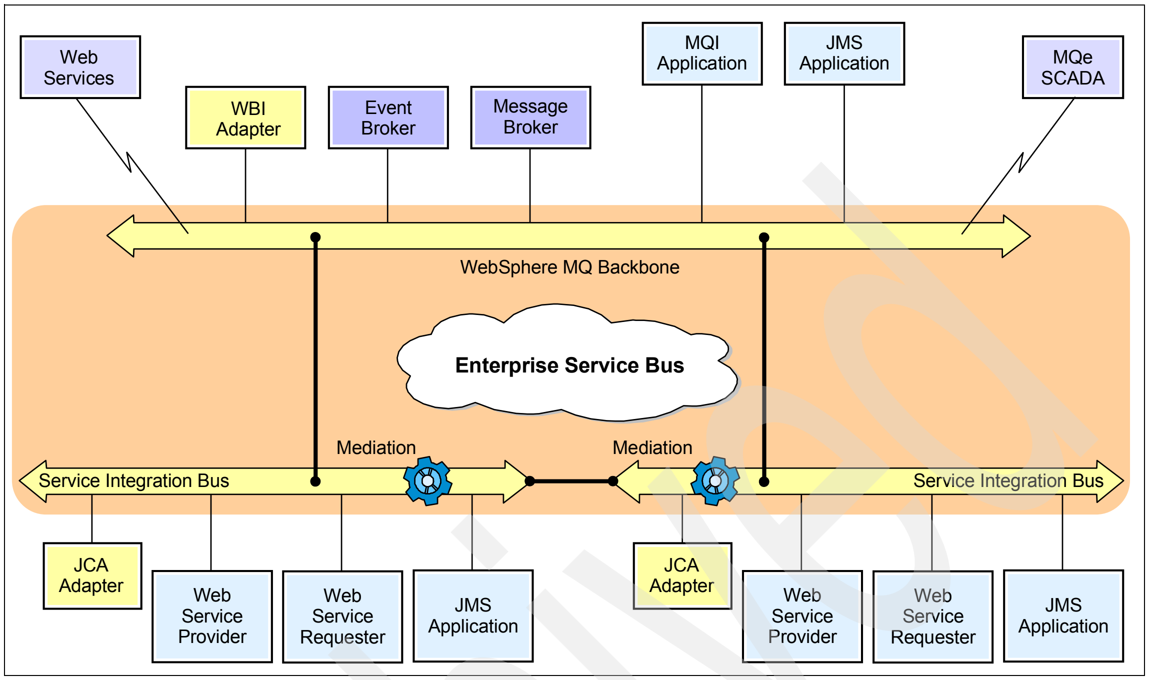


Figure 4-5 The Enterprise Service Bus

A service integration bus consists of the following:

- Bus members

  Application servers or clusters that have been added to the bus.

- Messaging engine

  The application server or cluster component that manages bus resources. When a bus member is defined, a messaging engine is automatically created on the application server or cluster. The messaging engine provides a connection point for clients to produce or consume messages from.

  An application server will have one messaging engine per bus it is a member of. A cluster will have at least one messaging engine per bus and can have more. In this case the cluster owns the messaging engine(s) and determines on which application server(s) they will run.

- Destinations

  The place within the bus that applications attach to exchange messages. Destinations can represent Web service endpoints, messaging point-to-point queues, or messaging publish/subscribe topics. Destinations are created on a bus and hosted on a messaging engine.

- Message store

  Each messaging engine uses a set of tables in a data store (JDBC database) to hold information such as messages, subscription information, and transaction states. Messaging engines can share a database, each using its own set of tables. The message store can be backed by any JDBC database supported by WebSphere Application Server.

4.10.1 Application support

The service integration bus supports the following application attachments:

- Web services
  - Requestors using the JAX-RPC API
  - Providers running in WebSphere Application Server as stateless session beans and servlets (JSR-109)
  - Requestors or providers attaching via SOAP/HTTP or SOAP/JMS
- Messaging applications
  - Inbound messaging using JFAP-TCP/IP (or wrapped in SSL for secure messaging). JFAP is a proprietary format and protocol used for service integration bus messaging providers.
  - MQ application in an MQ network using MQ channel protocol
  - JMS applications in WebSphere Application Server V5 using MQ client protocol
  - JMS applications in WebSphere Application Server V6

4.10.2 Service integration bus and messaging

With Express or WebSphere Application Server configuration, you will typically have one stand-alone server with one messaging engine on one service integration bus. With Network Deployment you have more flexibility in your options.

The following are valid topologies:

- One bus and one messaging engine (application server or cluster).
- One bus with multiple messaging engines.
- Multiple buses within a cell. They can or cannot be connected to each other.
- Buses connected between cells.
- One application server that is a member of multiple buses. It will have one messaging engine per bus.

- A connection between a bus and a WebSphere MQ queue manager. When using this type of topology, consider the following:
  - WebSphere MQ can coexist on the same machine as the WebSphere default messaging provider. (In V5, the embedded JMS server and WebSphere MQ could not coexist on the same machine.)
  - A messaging engine cannot participate in a WebSphere MQ cluster.
  - The messaging engine can be configured to look like another queue manager to WebSphere MQ.
  - WebSphere applications can send messages to WebSphere MQ queues direct or though the service integration bus.
  - You can have multiple connections to WebSphere MQ, but each must be to a different queue manager.
  - WebSphere Application Server V5 JMS client can connect to V6 destinations. Also, a V6 JMS application can connect to an embedded JMS provider in a V5 server if so configured. However, you cannot connect a V5 embedded JMS server to a V6 bus.

Mediation

Mediation is the ability to manipulate a message as it traverses the messaging bus (destination). For example:

- Transform the message
- Reroute the message
- Copy and route to additional destinations
- Interact with non-messaging resource managers (databases, for example)

Mediation is controlled using a mediation handler list. The list is a collection of mediation handlers (Java programs that perform the function of a mediation) that are invoked in sequence.

Clustering

In a distributed server environment, you can use clustering for high availability or scalability. As you have seen, a cluster can be added as a bus member.

- High availability

  One messaging engine is active in the cluster. In the event that the messaging engine or server fails, the messaging engine on a standby server is activated.

- Scalability

  A single messaging destination can be partitioned across multiple active messaging engines in the cluster (messaging order is not preserved).

Quality of Service

Quality of service (QoS) can be defined on a destination basis to determine how messages are (or are not) persisted. QoS can also be specified within the application.

Message-driven beans

With EJB 2.1, message-driven beans (MDBs) in the application server that listen to queues and topics are linked to the appropriate destinations on the service integration bus using JCA connectors (ActivationSpec objects). Support is also included for EJB 2.0 MDBs to be deployed against a listener port.

4.10.3 Web services and the integration bus

Through the service integration bus Web services enablement, you can achieve the following goals:

- You can take an internal service that is available at a service destination, and make it available as a Web service.
- You can take an external Web service and make it available at a service destination.
- You can use the Web services gateway to map an existing service — either an internal service or an external Web service — to a new Web service that appears to be provided by the gateway.

4.11 Security

Table 4-6 shows the security features that the WebSphere Application Server configurations support.

Table 4-6 WebSphere Application Server security support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Java 2 security	Yes	Yes
J2EE security (role mapping)	Yes	Yes
JAAS	Yes	Yes
CSIv2	Yes	Yes
JACC	Yes	Yes
Security authentication	LTPA, SWAM	LTPA
User registry	Local OS, LDAP, Custom Registry	Local OS, LDAP, Custom Registry

Figure 4-6 presents a general view of the logical layered security architecture model of WebSphere Application Server. The flexibility of that architecture model lies in pluggable modules that can be configured according to the requirements and existing IT resources.

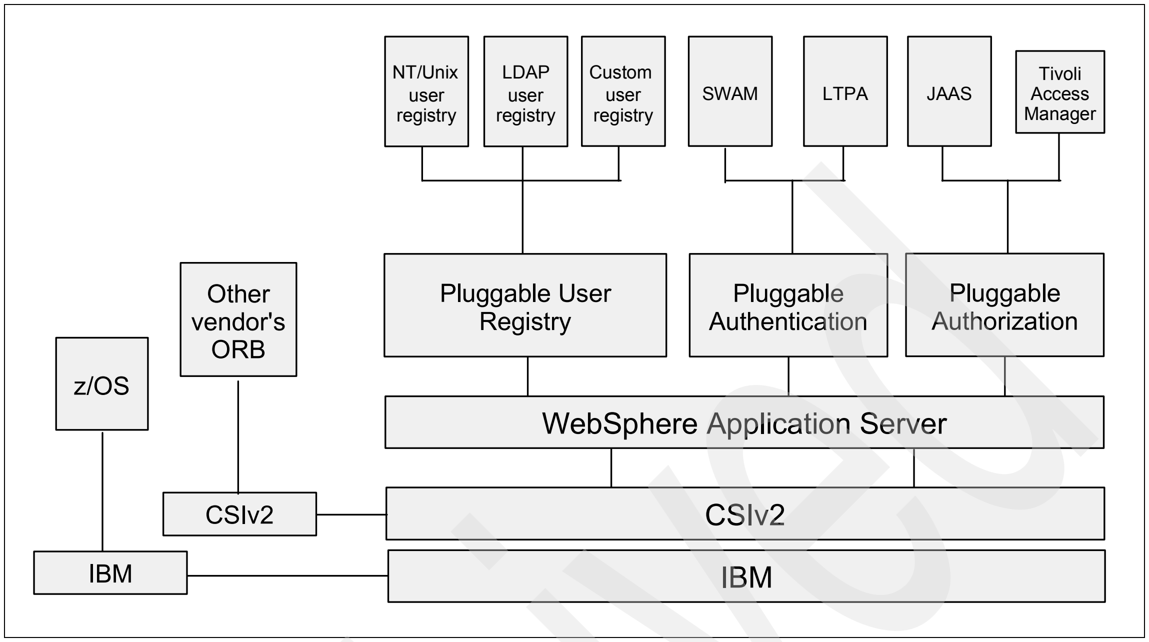


Figure 4-6 WebSphere Application Server security architecture

IBM WebSphere Application Server security sits on top of the operating system security and the security features provided by other components, including the Java language.

- Operating system security protects sensitive WebSphere configuration files and authenticates users when the operating system user registry is used for authentication.
- Standard Java security is provided through the JVM that is used by WebSphere and the Java security classes.
- The Java 2 Security API provides a means to enforce access control, based on the location of the code and who signed it. Java 2 security guards access to system resources such as file I/O, sockets, and properties. WebSphere global security settings allow you to enable or disable Java 2 security and provide a default set of policies. Java 2 security can be activated or inactivated independently from WebSphere global security.

  The current principal of the thread of execution is not considered in the Java 2 security authorization. There are instances where it is useful for the authorization to be based on the principal, rather the code base and the signer.

- The JAAS is a standard Java API that allows the Java 2 security authorization to be extended to the code base on the principal as well as the code base and signers. The JAAS programming model allows the developer to design application authentication in a pluggable fashion, which makes the application independent from the underlying authentication technology. JAAS does not require Java 2 security to be enabled.
- The Common Secure Interoperability protocol adds additional security features that enable interoperable authentication, delegation and privileges in a CORBA environment. It supports interoperability with the EJB 2.1 specification and can be used with SSL.
- J2EE security uses the security collaborator to enforce J2EE-based security policies and support J2EE security APIs. APIs are accessed from WebSphere applications in order to access security mechanisms and implement security policies. J2EE security guards access to Web resources such as servlets/JSPs and EJB methods based on roles defined by the application developer. Users and groups are assigned to these roles during application deployment.
- JACC support allows the use of third party authorization providers to be used for access decisions. The default JACC provider for WebSphere Application Server is the Tivoli Access Manager (bundled with Network Deployment). The Tivoli Access Manager client functions are integrated in WebSphere Application Server.
- IBM Java Secure Socket Extension (JSEE) is the Secure Sockets Layer (SSL) implementation used by WebSphere Application Server. It is a set of Java packages that enable secure Internet communications. It implements a Java version of SSLand Transport Layer Security (TLS) protocols and includes functionality for data encryption, server authentication, message integrity, and client authentication.

WebSphere Application Server security relies on and enhances all the above mentioned layers. It implements security policy in a unified manner for both Web and EJB resources. WebSphere global security options are defined at the cell level although individual servers can override a subset of the security configuration. When using mixed z/OS and distributed nodes, the security domain features will be merged.

4.11.1 User registry

The pluggable user registry allows you to configure different databases to store user IDs and passwords that are used for authentication and authorization. There are three options:

- Local operating system user registry

  When configured, WebSphere uses the operating system's users and groups for authentication.

- LDAP user registry

  In many solutions, LDAP user registry is recommended as the best solution for large scale Web implementations. Most of the LDAP servers available on the market are well equipped with security mechanisms that can be used to securely communicate with WebSphere Application Server. The flexibility of search parameters that an administrator can set up to adapt WebSphere to different LDAP schemas is considerable.

- Custom user registry

  The custom user registry leaves an open door for any custom implementation of a user registry database. WebSphere API provides the UserRegistry Java interface that you should use to write the custom registry. This interface can be used to access virtually any relational database, flat files, and so on.

Only one single registry can be active at a time.

4.11.2 Authentication

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application. The pluggable authentication module allows you to choose whether WebSphere authenticates the user or accepts the credentials from external authentication mechanisms.

An authentication mechanism in WebSphere typically collaborates closely with a user registry when performing authentication. The authentication mechanism is responsible for creating a credential which is a WebSphere internal representation of a successfully authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although WebSphere provides several authentication mechanisms, only a single active authentication mechanism can be configured at once. The active authentication mechanism is selected when configuring WebSphere global security.

WebSphere provides two authentication mechanisms: Simple WebSphere Authentication Mechanism and Lightweight Third Party Authentication. These two authentication mechanisms differ primarily in the distributed security features each supports.

- Simple WebSphere Authentication Mechanism

  The SWAM authentication mechanism is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction is due to the fact that SWAM does not support forwarding of credentials. Thus, if a servlet or EJB in application server process 1 invokes a remote method on an EJB living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, might cause authorization failures.

  Because SWAM is intended for a single application server process, single signon (SSO) is not supported.

  The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

  SWAM relies on the session ID. It is not as secure as LTPA, therefore using SSL with SWAM is strongly recommended.

- Lightweight Third Party Authentication

  Lightweight Third Party Authentication is intended for distributed, multiple application servers and machine environments. It supports forwarding of credentials and SSO. LTPA is able to support security in a distributed environment through the use of cryptography. This allows LTPA to encrypt, digitally sign and securely transmit authentication related data and later decrypt and verify the signature.

  LTPA requires that the configured user registry be a central shared repository such as LDAP or a Windows domain type registry.

4.11.3 Authorization

WebSphere Application Server standard authorization features:

- Java 2 security architecture, which uses a security policy to specify who is allowed to execute code in the application. Code characteristics, like a code signature, signer ID, or source server, determine whether or not the code will be granted access to be executed.
- JAAS, which extends this approach with role-based access control. Permission to execute a code is granted not only based on the code

characteristics but also on the user running it. JAAS programming models allow the developer to design application authentication in a pluggable fashion, which makes the application independent from the underlying authentication technology.

For each authenticated user, a Subject class is created and a set of Principals is included in the subject in order to identify that user. Security policies are granted based on possessed principals.

WebSphere Application Server provides an internal authorization mechanism that is used by default. As an alternative, you can define external JACC providers to handle authorization decisions. During application installation, security policy information is stored in the JACC provider server using standard interfaces defined by JACC. Subsequent authorization decisions are made using this policy information. As an exception to this, all administrative security authorization decisions are made by the WebSphere Application Server default authorization engine.

4.11.4 Security components

Figure 4-7 on page 125 shows an overview of the security components that come into play in WebSphere Application Security.

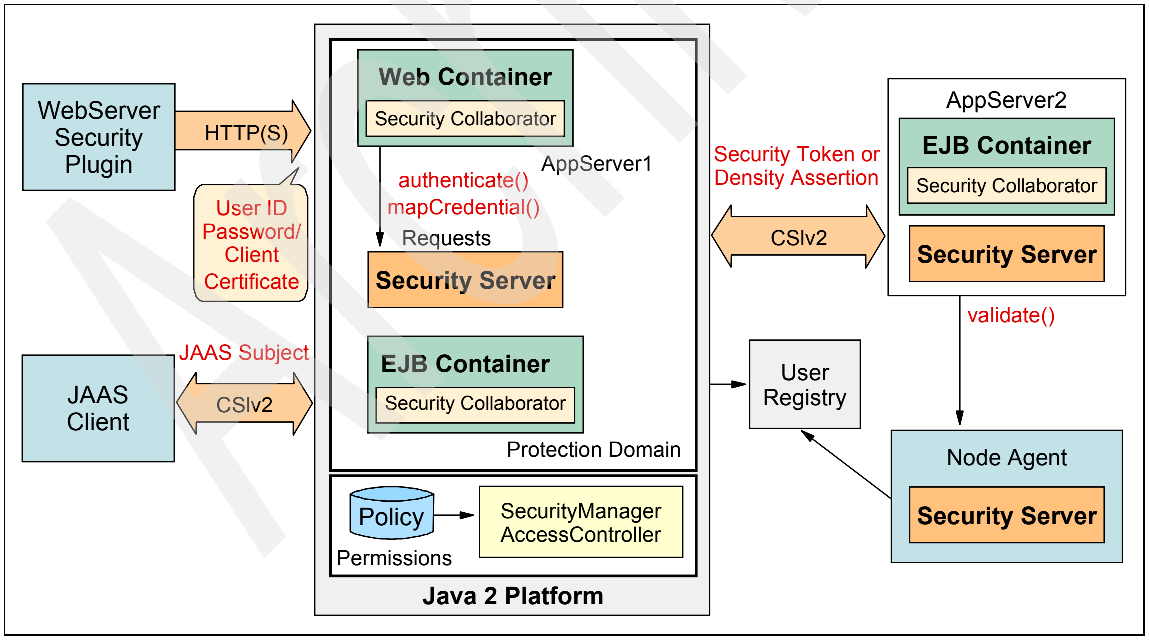


Figure 4-7 WebSphere Application Security components

Security server

The security server is a component of WebSphere Application Server that runs in each application server process. If multiple application server instances are executed on a single node, then multiple security servers exist on that node.

The security server component is responsible for managing authentication and for collaborating with the authorization engine and the user registry.

Security collaborators

Security collaborators are application server processes that enforce security constraints that are specified in deployment descriptors. They communicate with the security server every time that authentication and authorization actions are required. The following security collaborators are identified:

- Web security collaborator

  The Web security collaborator resides in the Web container and provides the following services to the application:

  – Checks authentication

  – Performs authorization according to the constraint that is specified in the deployment descriptor

  – Logs security tracing information

- EJB security collaborator

  The EJB security collaborator resides in the EJB container. It uses CSIv2 and SAS to authenticate Java client requests to enterprise beans. It works with the security server to perform the following functions:

  – Checks authorizations according to the specified security constraint

  – Supports communication with local user registry

  – Logs security tracing information

  – Communicates with external ORB using CSIv2 when a request for a remote bean is issued

4.11.5 Security flows

The following sections outline the general security flow.

Web browser communication

The following steps describe the interaction of the components from a security point of view when a Web browser sends a request to a WebSphere application.

1. The Web user requests a Web resource that is protected by WebSphere Application Server.

2. The Web server receives the request and recognizes that the requested resource is on the application server, and using the Web server plug-in, it redirects the request.
3. The Web server plug-in passes the user credentials to the Web security collaborator, which performs user authentication.
4. After successful authentication, the Web request reaches the Web container. The Web security collaborator passes the user's credentials and the security information contained in the deployment descriptor to the security server for authorization.
5. Upon subsequent requests, authorization checks are performed either by the Web collaborator or the EJB collaborator, depending on what the user is requesting. User credentials are extracted from the established security context.

Administrative tasks

Administrative tasks are issued using either the Web-based administrative console or the wsadmin scripting tool. The following steps illustrate how the administration tasks are executed:

1. The administration client generates a request that reaches the server side ORB and JMX MBeans. The JMX MBeans represent managed resources.
2. The JMX MBeans contact the security server for authentication purposes. JMX beans have dedicated roles assigned and do not use user registry for authentication and authorization.

Java client communication

The following steps describe how a Java client interacts with a WebSphere application:

1. A Java client generates a request that reaches the server side ORB.
2. The CSIv2 or IBM SAS interceptor performs authentication on the server side on behalf of the ORB and sets the security context.
3. The server side ORB passes the request to the EJB container.
4. After submitting a request to the access protected EJB method, the EJB container passes the request to the EJB collaborator.
5. The EJB collaborator reads the deployment descriptor from the .ear file and user credential from the security context.
6. Credentials and security information are passed to the security server which validates user access rights and passes this information back to the collaborator.

7. After receiving a response from the security server, the EJB collaborator authorizes or denies access to the requested resource.

4.12 Resource providers

Resource providers define resources that are needed by running J2EE applications. Table 4-7 shows the resource provider support of the WebSphere Application Server configuration.

Table 4-7 WebSphere Application Server resource provider support

	WebSphere Application Server - Express and WebSphere Application Server	WebSphere Application Server Network Deployment
JDBC provider	Yes	Yes
Mail providers (JavaMail)	Yes	Yes
JMS providers	Yes	Yes
Resource environment providers	Yes	Yes
URL providers	Yes	Yes
Resource adapters	Yes	Yes

4.12.1 JDBC resources

A data source represents a real-world data source, such as a relational database. When a data source object has been registered with a JNDI naming service, an application can retrieve it from the naming service and use it to make a connection to the data source it represents.

Information about the data source and how to locate it, such as its name, the server on which it resides, its port number, and so on, is stored in the form of properties on the DataSource object. This makes an application more portable because it does not need to hard code a driver name, which often includes the name of a particular vendor. It also makes maintaining the code easier because, for example, if the data source is moved to a different server, all that needs to be done is to update the relevant property in the data source. None of the code using that data source needs to be touched.

When a data source has been registered with an application server's JNDI name space, application programmers can use it to make a connection to the data source it represents.

The connection will usually be a pooled connection. That is, when the application closes the connection, the connection is returned to a connection pool, rather than being destroyed.

Data source classes and JDBC drivers are implemented by the data source vendor. By configuring a JDBC provider, we provide information about the set of classes that are used to implement the data source and the database driver. That is, it provides the environment settings for the DataSource object.

Data sources

In WebSphere Application Server, connection pooling is provided by two parts: a JCA Connection Manager and a relational resource adapter.

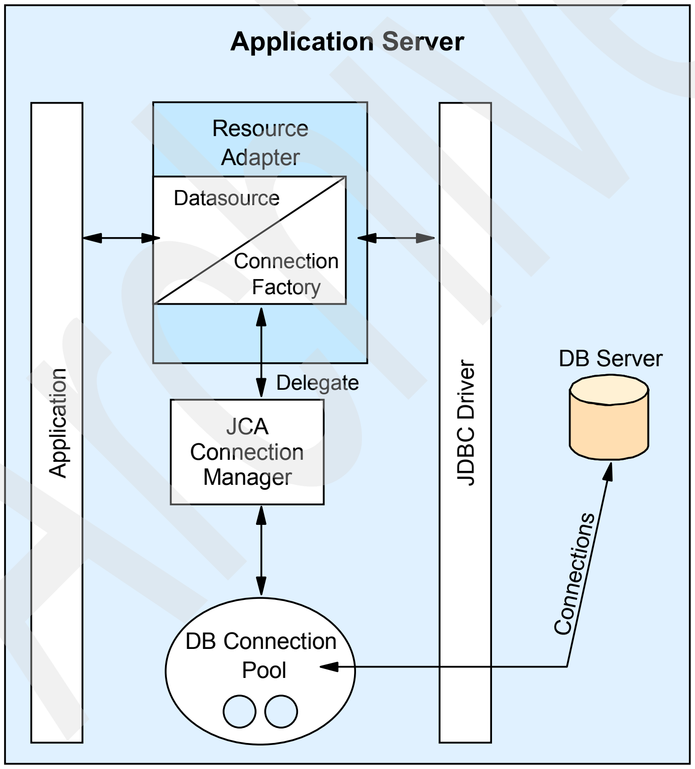


Figure 4-8 Resource adapter in J2EE connector architecture

The JCA Connection Manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides both JDBC wrappers and JCA CCI implementation that allow BMP, JDBC applications and container-managed persistence (CMP) beans to access the database.

Version 4 data sources: WebSphere Version 4.0 provided its own JDBC connection manager to handle connection pooling and JDBC access. This support is included with WebSphere Application Server V5 and V6 to provide support for J2EE 1.2 applications. If an application chooses to use a Version 4 data source, the application will have the same connection behavior as in WebSphere Version 4.

4.12.2 Mail providers

The JavaMail APIs provide a platform and protocol-independent framework for building Java-based mail client applications. The JavaMail APIs require service providers, known in WebSphere as protocol providers, to interact with mail servers that run the pertaining protocols.

A mail provider encapsulates a collection of protocol providers. WebSphere Application Server has a Built-in Mail Provider that encompasses three protocol providers: SMTP, POP3, and IMAP. These protocol providers are installed as the default and should be sufficient for most applications.

- Simple Mail Transfer Protocol (SMTP). SMTP is a popular transport protocol for sending mail. JavaMail applications can connect to an SMTP server and send mail through it by using this SMTP protocol provider.
- Post Office Protocol (POP3). POP3 is the standard protocol for receiving mail.
- Internet Message Access Protocol (IMAP). IMAP is an alternative protocol to POP3 for receiving mail.

To use other protocols, you must install the appropriate service provider for those protocols.

In addition to service providers, JavaMail requires the JavaBeans Activation Framework (JAF) as the underlying framework to deal with complex data types that are not plain text, such as Multipurpose Internet Mail Extensions (MIME), Uniform Resource Locator (URL) pages, and file attachments.

The JavaMail APIs, the JAF, the service providers, and the protocols are shipped as part of WebSphere Application Server using the following Sun licensed packages:

- mail.jar file, which contains the JavaMail APIs, and the SMTP, IMAP, and POP3 service providers.
- activation.jar file, which contains the JAF.

4.12.3 JCA resource adapters

JCA defines a standard architecture for connecting the J2EE platform to heterogeneous EIS, for example ERP, mainframe transaction processing, database systems, and legacy applications that are not written in the Java programming language.

The JCA resource adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the connectivity between J2EE components (an application server or an application client) and an EIS.

To use a resource adapter, you need to install the resource adapter code and create connection factories that use the adapter.

One resource adapter, the WebSphere Relational Resource Adapter, is predefined for handling data access to relational databases. This resource adapter provides data access through JDBC calls to access databases dynamically. It provides connection pooling, local transaction, and security support. The WebSphere persistence manager uses this adapter to access data for CMP beans.

4.12.4 URL providers

URL providers implement the functionality for a particular URL protocol, such as HTTP, by extending the java.net.URLStreamHandler and java.net.URLConnection classes. It enables communication between the application and a URL resource that is served by that particular protocol.

A URL provider named Default URL Provider is included in the initial WebSphere configuration. This provider utilizes the URL support provided by the IBM JDK. Any URL resource with protocols based on the Java 2 Standard Edition 1.3.1, such as HTTP, FTP or File, can use the default URL provider.

You can also plug in your own URL providers that implement other protocols that are not supported by the JDK.

4.12.5 JMS providers

The JMS functionality that WebSphere provides includes support for the following types of JMS provider:

- Default messaging provider (service integration bus)
- WebSphere MQ provider
- Generic JMS providers
- V5 default messaging provider (for migration)

There can be more than one JMS provider per node. That is, a node can be configured to concurrently make use of any combination (or all) of the default messaging provider, WebSphere MQ JMS provider and a generic JMS provider. In addition, WebSphere MQ and the default messaging provider can coexist on the same machine.

The support that WebSphere administration tools provides for configuration of JMS providers differs depending upon the provider. Table 4-8 provides a summary of the support.

Table 4-8 WebSphere administration support for JMS provider configuration

Configurable objects	Default messaging provider	WebSphere MQ JMS provider	Generic JMS provider	V5 default messaging, WebSphere JMS provider
Messaging system objects (queues/topics)	Yes	No	No	Yes
JMS administered objects (JMS connection factory and JMS destination)	Yes	Yes	No	Yes

Default messaging provider

The default messaging provider for WebSphere Application Server is the service integration bus, which provides both point-to-point and publish/subscribe functions. Within this provider, you define JMS connection factories and JMS destinations corresponding to service integration bus destinations.

WebSphere MQ JMS provider

WebSphere Application Server supports the use of full WebSphere MQ as the JMS provider. The product is tightly integrated with the WebSphere installation, with WebSphere providing the JMS client classes and administration interface, while WebSphere MQ provides the queue-based messaging system.

Generic JMS providers

WebSphere Application Server supports the use of generic JMS providers, as long as they implement the ASF component of the JMS 1.0.2 specification. JMS resources for generic JMS providers are not configurable using WebSphere administration.

V5 default JMS provider

For backwards compatibility with earlier releases, WebSphere Application Server also includes support for the V5 default messaging provider which enables you to configure resources for use with V5 embedded messaging. The V5 default messaging provider can also be used with a service integration bus.

4.12.6 Resource environment providers

The java:comp/env environment provides a single mechanism by which both JNDI name space objects and local application environment objects can be looked up. WebSphere Application Server provides a number of local environment entries by default.

The J2EE specification also provides a mechanism for defining custom (non-default) environment entries using <resource-env-ref> entries defined in an application's standard deployment descriptors. The J2EE specification separates the definition of the resource environment entry from the application by:

1. Requiring the application server to provide a mechanism for defining separate administrative objects that encapsulate a resource environment entry. The administrative objects are to be accessible via JNDI in the application server's local name space (java:comp/env).
2. Specifying the administrative object's JNDI lookup name and the expected returned object type in <resource-env-ref>.

WebSphere Application Server supports the <resource-env-ref> mechanism by providing administration objects for the following:

- Resource environment provider

  Defines an administrative object that groups together the referenceable and resource environment entry administrative objects as well as any required custom properties.
- Referenceable

  Defines the class name of the factory class that returns object instances implementing a Java interface.

- Resource environment entry

  Defines the binding target (JNDI name), factory class and return object type (via the link to the referenceable) of the resource environment entry.

4.13 Workload management

Clustering application servers that host Web containers automatically enables plug-in workload management for the application servers and the servlets they host. Routing of servlet requests occurs between the Web server plug-in and the clustered application servers using HTTP or HTTPS.

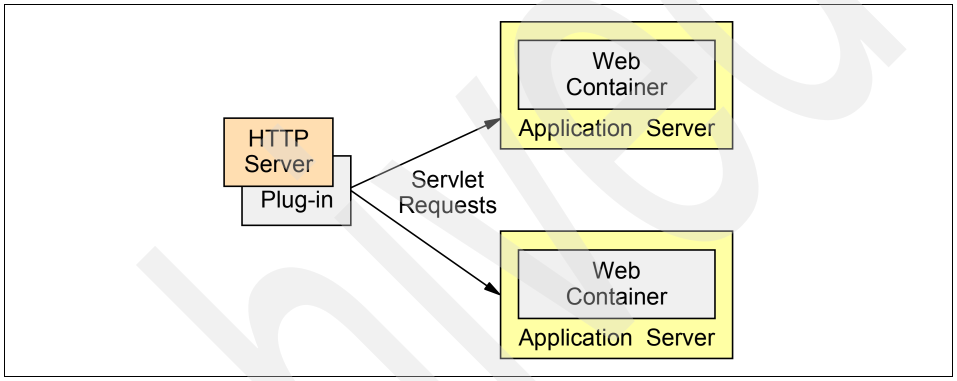


Figure 4-9 Plug-in (Web container) workload management

This routing is based on weights associated with the cluster members. If all cluster members have identical weights, the plug-in will send equal requests to all members of the cluster assuming no strong affinity configurations. If the weights are scaled in the range from zero to twenty, the plug-in will route requests to those cluster members with the higher weight value more often. A rule of thumb formula for determining routing preference would be:

% routed to Server1 = weight1 / (weight1+weight2+...+weight*n*)

where there are n cluster members in the cluster.

The Web server plug-in temporarily routes around unavailable cluster members.

Workload management for EJB containers can be performed by configuring the Web container and EJB containers on separate application servers. Multiple application servers with the EJB containers can be clustered, enabling the distribution of EJB requests between the EJB containers.

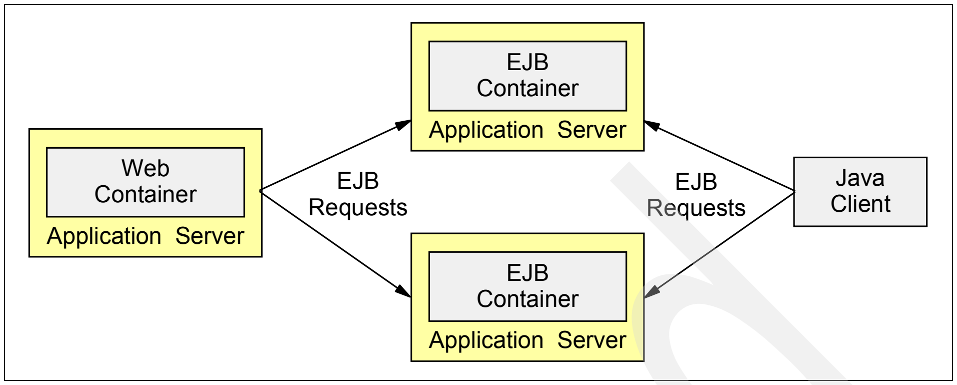


Figure 4-10   EJB workload management

In this configuration, EJB client requests are routed to available EJB containers in a round robin fashion based on assigned server weights. The EJB clients can be servlets operating within a Web container, stand-alone Java programs using RMI/IIOP or other EJBs.

The server-weighted round robin routing policy ensures a distribution that is based on the set of server weights that have been assigned to the members of a cluster. For example, if all servers in the cluster have the same weight, the expected distribution for the cluster would be that all servers receive the same number of requests. If the weights for the servers are not equal, the distribution mechanism will send more requests to the higher weight value servers than the lower weight value servers. The policy ensures the desired distribution, based on the weights assigned to the cluster members. In V6, the balancing mechanism for weighted round-robin had been enhanced to ensure more balanced routing distribution among servers.

You can also choose to have requests sent to the node on which the client resides as the preferred routing. In this case, only cluster members on that node will be chosen (using the round robin weight method). Cluster members on remote nodes will only be chosen if a local server is not available.

4.14 High availability

With Network Deployment V6, the high availability features have been significantly improved. The following is a quick overview of the failover capabilities:

- HTTP server failover

  The use of multiple HTTP servers, along with a load balancing product such as provided with the Edge components can be used to provide HTTP Server failover.

- Web container failover

  The HTTP server plug-in in the Web server is aware of the configuration of all Web containers and can route around a failed Web container in a cluster. Sessions can be persisted to a database or in-memory using data replication services.

- EJB container failover

  Client code and the ORB plug-in can route to the next EJB container in the cluster.

- Deployment manager and node agent

  The need for failover in these two components has been reduced, thus no built-in failover capability has been provided. The loss of the deployment manager only affects configuration. It is recommended that you use a process nanny to restart the node agent if it fails.

- Critical services failover:

  Hot standby and peer failover for critical services (WLM routing, PMI aggregation, JMS messaging, transaction manager, and so forth) is provided through the use of HA domains.

  An HA domain defines a set of WebSphere processes (a core group) that participate in providing high availability function to each other. Processes in the core group can be the deployment manager, node agents, application servers, or cluster members.

  One or more members of the core group can act as an HA coordinator, managing the HA activities within the core group processes. If an HA coordinator server fails, another server in the core group will take over the HA coordinator duties. HA policies defined how the failover occurs.

  Workload management information is shared between core members and failover of critical services is done among them in peer-to-peer fashion. Little configuration is necessary and in many cases, this function works with the defaults that are created automatically as you create the processes.

- Transaction log hot standby

  With V6, transaction logs can be maintained on Network Attached Storage. When a cluster member fails, another cluster member recovers the transaction log, thus enabling the failover 2PC transactions. The time required for the failover is dramatically reduced from what it took in V5.

- JMS messaging failover

  The messaging engine keeps messages in a remote database. When a server in a cluster fails, WebSphere selects an online server to run the messaging engine and the workload manager routes JMS connections to that server.

4.15 Administration

WebSphere Application Server's administration model is based on the JMX framework. JMX allows you to wrap hardware and software resources in Java and expose them in a distributed environment. JMX also provides a mapping framework for integrating existing management protocols, such as SNMP, into JMX's own management structures.

Each application server has an administration service that provides the necessary functions to manipulate configuration data for the server and its components. The configuration is stored in a repository. The repository is actually a set of XML files stored in the server's file system.

4.15.1 Administration tools

Table 4-9 shows the administration tool support for WebSphere Application Server by configuration.

Table 4-9 WebSphere Application Server administration tool support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Administrative console	Yes	Yes
Commands	Yes	Yes
wsadmin	Yes	Yes

Administrative console

The administrative console is a Web-browser based interface that provides configuration and operation capability. The administrator connects to the application using a Web browser client. Users assigned to different administration roles can manage the application server and certain components and services using this interface.

The administrative console is a system application, crucial to the operation of WebSphere and as such, is not exposed as an enterprise application on the console. In stand-alone application servers, the administrative console runs in the application server. In the Network Deployment distributed server environment, the administrative console application runs on the deployment manager. When a node is added to a cell, the administrative console application is deleted from the node and the configuration files are integrated into the master cell repository to be maintained by the deployment manager.

Commands

WebSphere Application Server provides a set of commands in the *<server_install>*/bin directory that allow you to perform a subset of administrative functions. For example, the **`startServer`** command is provided to start an application server.

The wsadmin scripting client

The wsadmin scripting client provides extra flexibility over the Web-based administration application, allowing administration using the command-line interface. Using the scripting client not only makes administration quicker, but helps automate the administration of multiple application servers and nodes using scripts.

The scripting client uses the Bean Scripting Framework (BSF), which allows a variety of scripting languages to be used for configuration and control. Two languages have been tested and are supported in V6: jacl and jython (or jpython).

The wsadmin scripting interface is included in all WebSphere Application Server configurations but is targeted toward advanced users. The use of wsadmin requires in-depth familiarity with application server architecture and a scripting language.

4.15.2 Configuration repository

The configuration repository holds copies of the individual component configuration documents stored in XML files. The application server's administrative service takes care of the configuration and makes sure it is consistent during the runtime.

The configuration of unfederated nodes can be archived for export and import, making them portable among different WebSphere Application Server instances.

4.15.3 Centralized administration

The Network Deployment package allows multiple servers and nodes to be administered from a central location. This is facilitated through the use of a central deployment manager that handles the administration process and distributes the updated configuration to the node agent for each node. The node agent, in turn, is responsible for maintaining the configuration for the servers in the node.

Table 4-10 WebSphere Application Server distributed administration support

	WebSphere Application Server - Express and WebSphere Application Server	**WebSphere Application Server Network Deployment**
Deployment manager	No	Yes
Node agent	No	Yes
Application servers	Stand-alone	Stand-alone or distributed server, clustering

Managed processes

All operating system processes that are components of the WebSphere product are called managed servers or managed processes. JMX support is embedded in all managed processes and these processes are available to receive administration commands and to output administration information about the state of the managed resources within the processes.

WebSphere provides the following managed servers and processes:

- deployment manager

  Provides a single point to access all configuration information and control for a cell. The deployment manager aggregates and communicates with all of the node agent processes on each node in the system.

- Node agent

  Aggregates and controls all of the WebSphere managed processes on its node. There is one node agent per node.

- Application server

  Managed server that hosts J2EE applications.

Deployment manager

The deployment manager process provides a single, central point of administrative control for all elements in the cell. It hosts the Web-based administrative console application. Administrative tools that need to access any managed resource in a cell usually connect to the deployment manager as the central point of control.

The deployment manager is responsible for the content of the repositories (configuration and application binaries) on each of the nodes. It manages the repositories through communication with the node agent process resident on each node of the cell.

Using the deployment manager, horizontal scaling, vertical scaling, and distributed applications are all easy to administer and manage, because application servers are managed by nodes, and one or more nodes are managed by a cell.

Node agent

The node agent is an administrative process and is not involved in application serving functions. It hosts important administrative functions such as:

- File transfer services
- Configuration synchronization
- Performance monitoring

The node agent aggregates and controls all the managed processes on its node by communicating with:

- The deployment manager to coordinate configuration synchronization and to perform management operations on behalf of the deployment manager.
- Application servers and managed Web servers to manage (start and stop) each server and to update its configuration and application binaries as required.

Only one node agent is defined (and run) on each node. In a stand-alone server environment, there is no node agent.

Master configuration repository

In a distributed server environment, a master configuration repository that contains all of the cell's configuration data is maintained by the deployment manager. The configuration repository at each node is a synchronized subset of the master repository. The node repositories are read-only for application server access, because only the deployment manager can initiate their update, pushing configuration changes out from the cell master configuration repository.

4.16 The flow of an application

Figure 4-11 shows the typical application flow for Web browser clients using either JDBC (from a servlet) or EJB to access application databases.

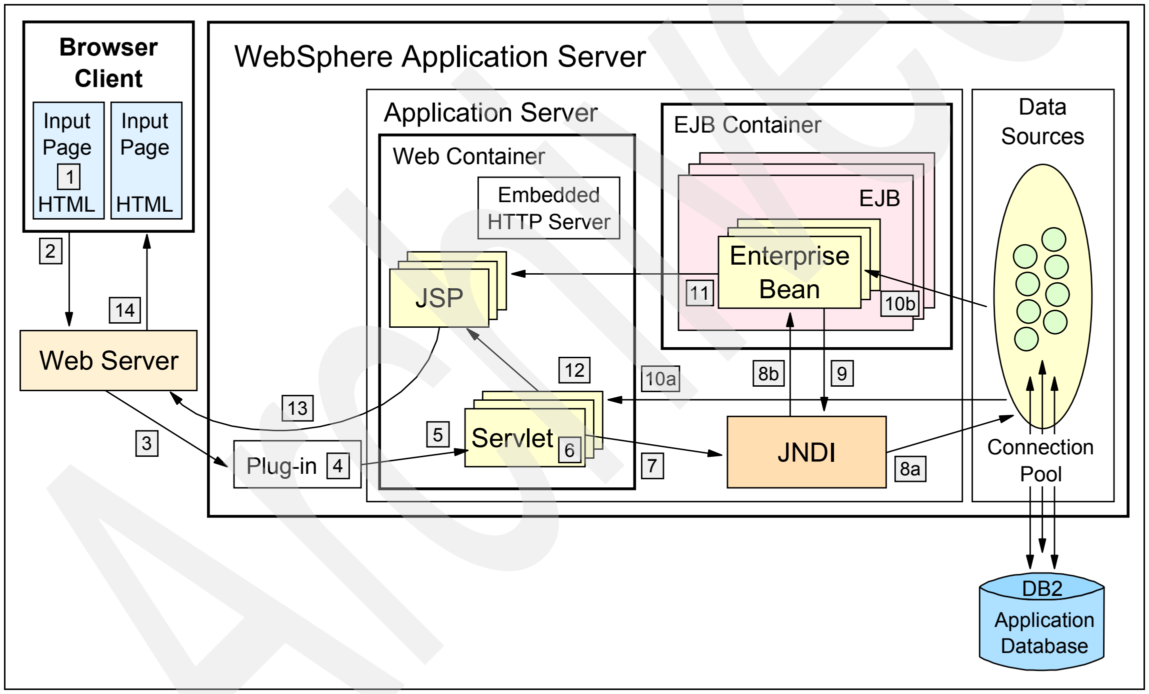


Figure 4-11 Application flow

A typical application flow is:

1. A Web client requests a URL in the browser (input page).
2. The request is routed to the Web server over the Internet.
3. The Web server immediately passes the request to the Web server plug-in. All requests go to the WebSphere plug-in first.

4. The Web server plug-in examines the URL, verifies the list of host name aliases from which it will accept traffic based on the virtual host information, and chooses a server to handle the request.
5. A stream is created. A stream is a connection to the Web container. It is possible to maintain a connection (stream) over a number of requests. The Web container receives the request and, based on the URL, dispatches it to the proper servlet.
6. If the servlet class is not loaded, the dynamic class loader loads the servlet (servlet init(), then doGet() or doPost()).
7. JNDI is now used for lookup of either datasources or EJBs required by the servlet.
8. Depending upon whether a datasource is specified or an EJB is requested, the JNDI directs the servlet to the corresponding database and gets a connection from its connection pool in the case of a data source.
9. To the corresponding EJB container, which then instantiates the EJB when an EJB is requested.
10. If the EJB that is requested involves an SQL transaction, it goes back to the JNDI to look up the datasource.
11. The SQL statement is executed and the data retrieved is sent back:
    - To the servlet
    - To the EJB
12. Data beans are created and handed off to JSPs in the case of EJBs.
13. The servlet sends data to JSPs.
14. The JSP generates the HTML that is sent back through the WebSphere plug-in to the Web server.
15. The Web server sends the output page (output HTML) to the browser.

4.17 Developing and deploying applications

Figure 4-12 shows a high level view of the stages of application development and deployment.

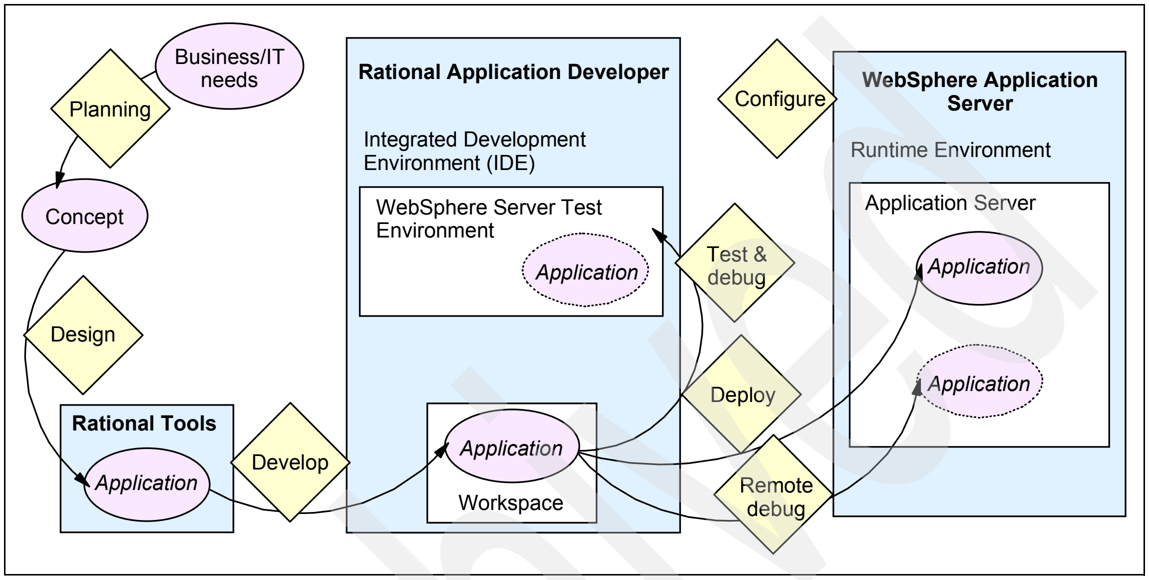


Figure 4-12 Develop and deploy

4.17.1 Application design

Design tools like Rational Rose® or Rational XDE™ can be used to model the application using the Unified Modeling Language (UML). The output of the modeling generally consists of use case scenarios, class diagrams, and starter code generated that is based on the model.

4.17.2 Application development

Application development is done using Rational Application Developer (or a comparable IDE) to create the enterprise application. You can start by importing pre-generated code such as from Rational Rose, a sample application, or an existing production application, or you can start from scratch.

Rational Application Developer provides many tools and aids to get you started quickly. It also supports team development using CVS or Rational ClearCase®, allowing multiple developers to share a single master source copy of the code.

During the development phase, component testing can be done using the built-in WebSphere Application Server test environment. Rational Application Developer provides server tools capable of creating and managing servers both in the test environment and on remote server installations. The application is packaged automatically into an EAR file for deployment when you run the application on a server using Rational Application Developer.

4.17.3 Application packaging

J2EE applications are packaged into Enterprise Application Archive (EAR) files to be deployed to one or more application servers. A J2EE application contains any or all of the modules shown in Table 4-11.

Table 4-11 J2EE 1.3 application modules

Module	Filename	Contents
Web module	<module>.war	Servlets, JSP files, and related code artifacts.
EJB module	<module>.jar	Enterprise beans and related code artifacts.
Application client module	<module>.jar	Application client code.
Resource adapter module	<module>.rar	Library implementation code that your application uses to connect to enterprise information systems (EIS).

This packaging is done automatically in Rational Application Developer when you export an application for deployment. If you are using another IDE, WebSphere Application Server (with the exception of the Express configuration) provides the Application Server Toolkit for packaging applications.

Enhanced EAR files

The enhanced EAR, introduced in WebSphere Application Server V6, is a regular J2EE EAR file with additional configuration information for resources usually required by J2EE applications. While adding this extra configuration information at packaging time is not mandatory, it can simplify deployment of J2EE applications to WebSphere.

When an enhanced EAR is deployed to a WebSphere Application Server V6 server, WebSphere can configure the resources specified in the enhanced EAR automatically. This configuration reduces the number of configuration steps required to set up the WebSphere environment to host the application.

4.17.4 Application deployment

Applications are installed on application servers using the administrative console or the wsadmin scripting interface. An application can be deployed to a single server or a cluster. In the case of a cluster, it is installed on each application server in the cluster.

Installing an application involves:

- Binding resource references (created during packaging) to actual resources. For example, a data source would need to be bound to a real database.
- Defining JNDI names for EJB home objects.
- Specifying data source entries for entity beans.
- Binding EJB references to the actual EJB JNDI names.
- Mapping Web modules to virtual hosts.
- Specifying listener ports for MDBs.
- Mapping application modules to application servers.
- Mapping security roles to users or groups.

The use of an enhanced EAR file simplifies this installation process.

After a new application is deployed, the Web server plug-in configuration file needs to be regenerated and copied to the Web server.

Application update

In previous releases, deploying an update to an application required a complete EAR file to be deployed and the application to be restarted. WebSphere Application Server V6 now allows partial updates to applications and makes it possible to restart only parts of an application.

Updates to an application can consist of individual application files, application modules, zip files containing application artifacts, or the complete application. All module types can be started, though only Web modules can be stopped.

In V6, you have a rollout start option for installing applications on a cluster that stops, updates, and starts each cluster member in turn, ensuring availability.

4.17.5 WebSphere Rapid Deployment

WebSphere Rapid Deployment is designed to simplify the development and deployment of WebSphere applications. It is a collection of Eclipse plug-ins that can be integrated within development tools or run in a headless mode from a user file system. WebSphere Rapid Deployment is currently integrated in

Rational Web Developer, Rational Application Developer, and the Application Server Toolkit. Initially, there are features that are only supported in headless mode. During development, annotation-based programming is used. The developer adds metadata tags into the application source code that are used to generate artifacts needed by the code, thus reducing the number of artifacts the developer needs to create.

These applications are packaged into an enhanced EAR file that contains the J2EE EAR file along with deployment information, application resources, and properties (environment variables, JAAS authentication entries, shared libraries, classloader settings, and JDBC resources). During installation, this information is used to create the necessary resources. Moving an application from one server to another also moves the resources.

WebSphere Rapid Deployment automates installation of applications and modules onto a running application server by monitoring the workspace for changes and then driving the deployment process.

4.18 Java 2 Platform, Enterprise Edition

The J2EE specification is the foundation of the application programming model for WebSphere Application Server. Many WebSphere extensions add value by extending the J2EE programming model, which this section briefly describes. Read this topic for a quick overview of key J2EE concepts, including the parts of the J2EE run-time environment and the J2EE application packaging and deployment.

The J2EE specification is the standard for developing, deploying, and running enterprise applications. For more details about the J2EE 1.4 specification, see:

http://java.sun.com/j2ee/1.4/docs/#specs

The following list is a quick reference to the primary J2EE 1.4 specifications:

- Java Servlet 2.4
- JavaServer Pages (JSP) 2.0
- Java Database Connectivity (JDBC) 3.0
- Enterprise JavaBeans (EJB) 2.1
- J2EE Connector Architecture (JCA) 1.5
- Java Message Service (JMS) 1.1
- Web Services for J2EE, Version 1.1

WebSphere Application Server V6 has completed the full J2EE certification test suite. The product supports all of the J2EE 1.4 APIs. You can check the list of J2EE-compatible products posted by Sun Microsystems at:

`http://java.sun.com/j2ee/compatibility.html`

For the related API documentation, see:

`http://java.sun.com/j2ee/1.4/docs/api/index.html`

J2EE defines a standard that applies to all aspects of designing, developing, and deploying multi-tier, server-based applications. The standard architecture that the J2EE specification defines is composed of the following elements:

- Standard application model for developing multi-tier applications.
- Standard platform for hosting applications.
- Compatibility test suite for verifying that J2EE platform products comply with the J2EE platform standard.
- Reference implementation providing an operational definition of the J2EE platform.

The J2EE platform specification describes the run-time environment for a J2EE application. This environment includes application components, containers, and resource manager drivers. The elements of this environment communicate with a specified set of standard services.

For further details about multi-tier runtime environments refer to Chapter 9, “Topologies” on page 247.

4.18.1 J2EE platform roles

The J2EE platform also defines a number of distinct roles that are performed during the application development and deployment life cycle:

- The product provider designs and offers the J2EE platform, APIs, and other features that are defined in the J2EE specification for purchase.
- The tool provider offers tools that are used for the development and packaging of application components as part of the J2EE specifications.
- The application component provider creates Web components, enterprise beans, applets, or application clients to use in J2EE applications.
- The application assembler takes a set of components that are developed by application component providers and assembles them in the form of an enterprise archive (EAR) file.

- The deployer is responsible for deploying an enterprise application into a specific operational environment that corresponds to a J2EE platform product.
- The system administrator is responsible for the operational environment on which the application runs.

Product providers and tool providers have a product focus. Application component providers and application assemblers focus on the application. Deployers and system administrators focus on providing the J2EE application with platform-specific artifacts, and on the platform run time.

These roles help to identify the tasks and people involved. Understanding this separation of roles is important because it helps to determine the approach when developing and deploying J2EE applications.

4.18.2 J2EE benefits

The J2EE specification provides customers a standard which can be used to ensure investment protection when purchasing or developing applications. Comprehensive, independent Compatibility Test Suites ensure vendor compliance with J2EE standards.

Some benefits of deploying to a J2EE-compliant architecture include:

- A simplified architecture that is based on standard components, services, and clients. The architecture maximizes the write-once, run-anywhere Java technology.
- Services providing integration with existing systems, including JDBC, JMS, JCA, Java IDL, the JavaMail API, and Java Transaction API (JTA and JTS) for reliable business transactions.
- Scalability to meet demand by distributing containers across multiple systems and using database connection pooling, for example.
- A better choice of application development tools and components from vendors providing standard solutions.
- A flexible security model that provides SSO support, integration with legacy security schemes, and a unified approach to securing application components.

The J2EE specifications are the result of an industry-wide effort that involves a large number of contributors. IBM has contributed in defining more than 80% of the J2EE APIs.

4.18.3 Application components and their containers

The J2EE programming model has four types of application components, which reside in these types of containers in the application server:

- Enterprise beans, which are run by the EJB container.
- Servlets and JavaServer Pages files, which are run by the Web container.
- Application clients, which are run by the application client container.

J2EE containers provide runtime support for application components. There must be one container for each application component type in a J2EE application. By having a container between the application components and the set of services, the J2EE specification can provide a federated view of the APIs for the application components.

A container provides the APIs to the application components that are used for accessing the services. The container can also handle security, resource pooling, state management as well as naming and transaction issues.

4.18.4 Standard services

The J2EE platform provides components with a set of standard services that they can use to interact with each other. See the following Sun Web page for descriptions of each standard service:

http://java.sun.com/products/

4.18.5 J2EE packaging

During a process called assembly, J2EE components are packaged into modules. Modules are then packaged into applications. Applications can be deployed on the application server. Each module and application contains a J2EE deployment descriptor. The deployment descriptor is an XML file that provides instructions for deploying the application.

4.19 Technology support summary

Table 4-12 on page 143 break down the highlights of support provided by each WebSphere Application Server packaging option.

Table 4-12 WebSphere Application Server features and technology support

	WebSphere Application Server - Express V6 and WebSphere Application Server V6	**WebSphere Application Server Network Deployment V6**
Client and server support for the Software Development Kit for Java Technology Edition 1.4 (SDK 1.4.2)	Yes	Yes
J2EE 1.2, 1.3 programming support	Yes	Yes
J2EE 14. programming support[1] ▶ EJB 2.1 ▶ Servlet 2.4 ▶ JSP 2.0 ▶ JMS 1.1 ▶ JTA 1.0 ▶ JavaMail 1.3 ▶ JAF 1.0 ▶ JAXP 1.2 ▶ Connector 1.5 ▶ Web Services 1.1 ▶ JAX-RPC 1.1 ▶ SAAJ 1.2 ▶ JAXR 1.0 ▶ J2EE Management 1.0 ▶ JMX 1.2 ▶ JACC 1.0 ▶ JDBC 3.0	Yes	Yes
WebSphere Rapid Deployment	Yes	Yes
Service Data Object (SDO)	Yes	Yes
Messaging support ▶ Integrated JMS 1.1 messaging provider ▶ Support for WebSphere MQ and generic JMS providers ▶ MDBs	Yes	Yes

	WebSphere Application Server - Express V6 and WebSphere Application Server V6	**WebSphere Application Server Network Deployment V6**
Web services runtime support	Yes	Yes
Security support ▶ Java 2 ▶ J2EE ▶ JACC 1.0 ▶ JAAS 1.0 ▶ CSIv2 and SAS authentication protocols ▶ LDAP or local operating system user registry[3] ▶ SWAM ▶ LTPA authentication mechanism ▶ Kerberos (Technology Preview)	Yes	Yes
Multi-node management and Edge components		
Workload management and failover	No	Yes
Deployment manager	No	Yes
Central administration of multiple nodes	No	Yes
Load Balancer		Yes
Caching Proxy		X
Dynamic caching	Yes	Yes
Performance and analysis tools		
Performance Monitoring Infrastructure (PMI)	Yes	Yes
Log Analyzer	Yes	Yes
Tivoli Performance Viewer (integrated in the administrative console)	Yes	Yes

	WebSphere Application Server - Express V6 and WebSphere Application Server V6	WebSphere Application Server Network Deployment V6
Administration and tools		
Administration and tools ► Web-based administrative console ► Integrated IBM HTTP Server and application server administrative console ► Administrative scripting ► Java Management Extension (JMX) 1.2 ► J2EE Management (JSR-077) ► J2EE Deployment (JSR-088) ► Application Server Toolkit	Yes	Yes
Web services ► JAX-RPC v1.0 for J2EE 1.3, v1.1 for J2EE 1.4 ► JSR-109 (Web services for J2EE) ► WS-I Basic Profile 1.1.2 support ► WS-I Simple SOAP Binding Profile 1.0.3 ► WS-I Attachments Profile 1.0 ► SAAJ 1.2 ► UDDI V2 and V3 ► JAXR ► WS-TX (transactions) ► SOAP 1.1 ► WSDL 1.1 for Web services ► WSIL 1.0 for Web services ► OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) ► OASIS Web Services Security: UsernameToken Profile 1.0 ► OASIS Web Services Security X.509 Certificate Token Profile	Yes	Yes
Web Services Gateway	No	Yes
Private UDDI v3 Registry	Yes	Yes

	WebSphere Application Server - Express V6 and WebSphere Application Server V6	WebSphere Application Server Network Deployment V6
Programming Model Extensions[2]		
► Last Participant Support ► Internationalization Service ► WorkArea Service ► ActivitySession Service ► Extended JTA Support ► Startup Beans ► Asynchronous Beans (now called WorkManager) ► Scheduler Service (now called Timer Service) ► Object Pools ► Dynamic Query ► Web Services Gateway Filter Programming Model (with migration support) ► Distributed Map ► Application Profiling	Yes	Yes
► Back-up Cluster Support ► Dynamic WLM	No	Yes

1. You can find the APIs that are required for J2EE 1.4 in the Application Programming Interface section of the J2EE 1.4 specifications at:
`http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf`
2. Business process choreography and business rule beans remain in WebSphere Business Integration Server Foundation.

5

Integration with other products

WebSphere Application Server V6 integrates with many other technologies. This chapter examines some of these technologies with a continuing emphasis on planning and design considerations for a WebSphere Application Server V6 environment. First, it looks at security and user registry integration, then at integration with business process integration technologies, and finally at integration with proprietary messaging software from IBM. It includes the following sections:

- Tivoli Access Manager
- IBM Directory Server
- WebSphere Business Integration
- MQ integration

5.1 Tivoli Access Manager

Tivoli Access Manager provides a more holistic security solution at the enterprise level than the standard security providing mechanisms found in WebSphere Application Server. The following sections give an overview of built-in WebSphere Application Server security, how WebSphere Application Server V6 integrates with Tivoli Access Manager, and when and why the two products might be used together.

5.1.1 WebSphere Application Server V6 security

WebSphere Application Server V6 provides its own security infrastructure. This infrastructure is composed of some mechanisms which are specific to WebSphere Application Server but also many that use open standards security technologies. The security technology that is used is widely proven, and the software can integrate with other enterprise technologies easily.

A brief overview of WebSphere security

The rich, standards based, architecture for WebSphere Application Server security offers various configuration options:

- At the most basic level, a single server can use the Simple WebSphere Authentication Mechanism (SWAM).
- If more than one server is required to share a security mechanism (that is if containers on more than one server need to be able to track user credentials across the servers) then Lightweight Third Party Authentication (LTPA) can be used. (It can also be used for single servers). To make this possible, LTPA generates a security token for authenticated users which is passed between servers.
- Reverse Proxy Servers (servers that mediate between Web clients and multiple servers behind a fire wall) can be integrated with LTPA in WebSphere Application Server to allow for client authentication. A trust association is implemented, which is a contract between the application server and Reverse Proxy Server. IBM WebSeal Reverse Proxy is such a reverse proxy product.
- An operating system, LDAP or customer **user registry** is configured to be the user registry for the environment. Only one registry can be configured at any one time.
- WebSphere uses Java Authentication and Authorization Service (JAAS) API, which enables services to authenticate and to enforce access controls upon users.

- WebSphere uses a specialized JAAS module to implement the user credentials mapping module in its J2EE Connector Architecture (J2C) implementation which allows WebSphere Application Server to integrate with other Enterprise Information Systems.
- Java 2 Security can also be enabled. This security mechanism which is part of the Java runtime, allows or disallows access for Java code to specific system resources based upon permissions which can be specified in a fine grained manner. For example, a Java application can be granted permission to access the operating system's file system for file input and output.

5.1.2 New security features in WebSphere Application Server V6

WebSphere Application Server V6 brings the following changes to its security features and implementation.

Java 1.4 relevant changes

WebSphere Application Server V6 uses Java SE and EE 1.4. In terms of security, this new specification is providing the following:

- Java Enterprise Edition 1.4 now prescribes the use of the JACC API specification. (The relevant Java Community Process Java Specification Request is JSR-115.) JACC allows application servers to interact with third-party authorization providers (such as Tivoli Access Manager) via standard interfaces to make authorization decisions. Previously, proprietary interfaces for third-party authorization providers had to be used.
- Java Standard Edition 1.4 integrates the JAAS API into the core Java packages, previously it was a separate optional API.

WebSphere Application Server - Express V6 and LTPA

WebSphere Application Server - Express V6 now supports LTPA. You can find more details about the built-in security mechanisms in WebSphere Application Server V6 in Chapter 17, "Planning for security" on page 487.

5.1.3 Tivoli Access Manager and WebSphere Application Server V6

The WebSphere Application Server V6 security infrastructure is in and of itself adequate for many situations and circumstances. However, integrating WebSphere Application Server with Tivoli Access Manager allows for a far more holistic, end-to-end integration of application security across the entire enterprise.

The advantages at the enterprise level of using this approach are:

- Reduced risk through a consistent services-based security architecture.
- Lower administration costs through centralized administration and fewer security subsystems.
- Faster development and deployment.
- Reduced application development costs because developers do not have to develop bespoke security subsystems.
- Built-in, centralized, and configurable handling of legislative business concerns such as privacy requirements.

Repositories

As with WebSphere Application Server security, Tivoli Access Manager requires a user repository. It supports many different repositories such as Microsoft Active Directory and iPlanet but ships with IBM SecureWay® LDAP Directory Server. Tivoli Access Manager can be configured to use the same user repository as WebSphere Application Server, allowing you to share user identities with both Access Manager and WebSphere Application Server.

Tivoli Access Manager Policy Server

The Access Manager Policy Server maintains the master authorization policy database which contains the security policy information for all resources and all credentials information of all participants in the secure domain, both users and servers. The authorization database is then replicated across all local authorization servers. IBM WebSeal Reverse Proxy Server, for example, has its own local authorization server.

Access Manager for WebSphere component

The Tivoli Access Manager clients are embedded in WebSphere Application Server V6. Tivoli Access Manager server is bundled with WebSphere Application Server Network Deployment V6 and higher versions. The Tivoli Access Manager client can be configured using the scripting and GUI management facilities of WebSphere Application Server V6. Tivoli Access Manager further integrates with WebSphere Application Server in that it supports the special subjects AllAuthenticated and Everyone special subjects.

Note: AllAuthenticated and Everyone are subjects that are specific to WebSphere Application Server. These special categories are where access to a resource can be granted to all those users who have been authenticated regardless of what repository user groups they might belong to and where access can be granted to all users whether or not they are authenticated.

All communication between the Tivoli Access Manager clients and the Tivoli Access Manager server are done via the new JACC API (as WebSphere Application Server V6 supports J2EE 1.4).

Figure 5-1 shows the integration interfaces between WebSphere Application Server V6 and Tivoli Access Manager.

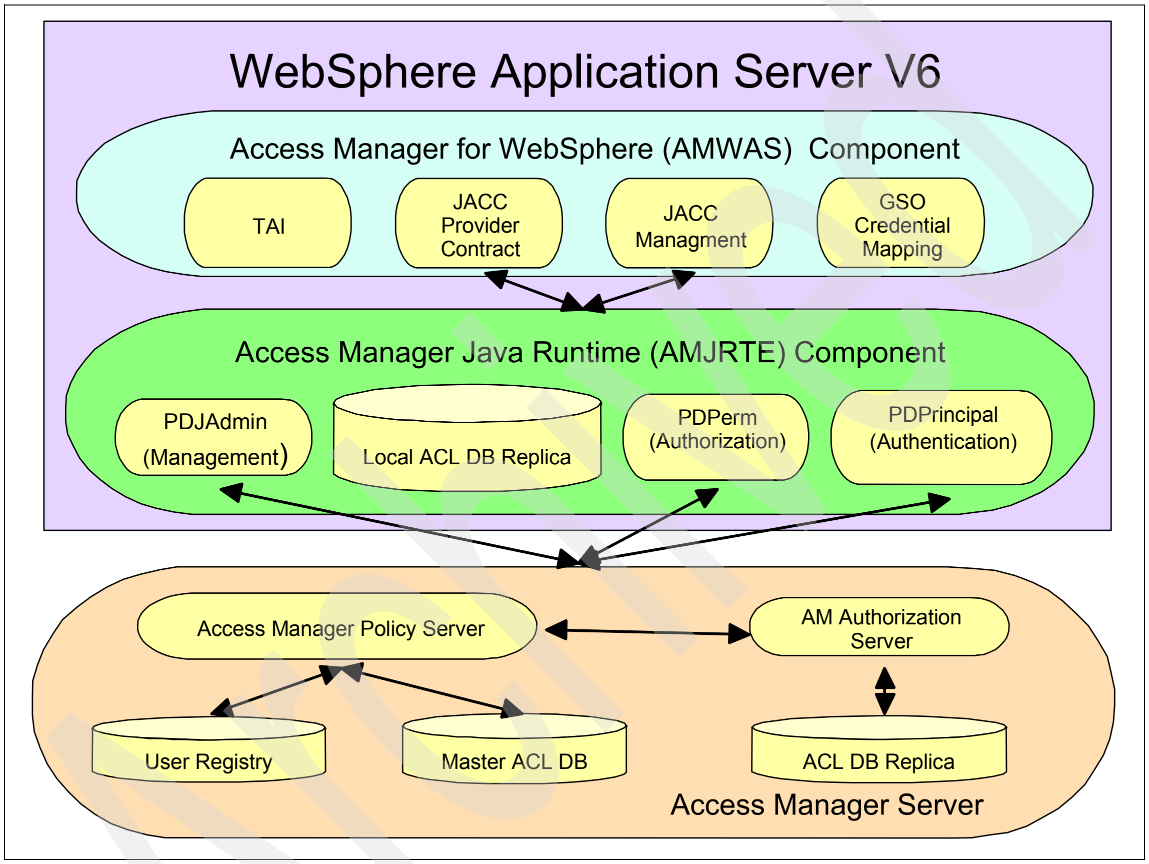


Figure 5-1 Integration of WebSphere Application Server V6 with Tivoli Access Manager

Further advantages of using Tivoli Access Manager

We have already reviewed the enterprise level advantages of using Tivoli Access Manager. Using Tivoli Access Manager at the application server level has the following further advantages:

- Provides industry leading security.
- Supports accounts and password policies.
- Supports dynamic changes to the authorization table without having to restart the applications.
- Provides tight integration with WebSphere Application Server.

Future enhancements

With the introduction of JACC to the Java Enterprise Edition 1.4 programming platform, integration of third-party security providers, especially Tivoli Access Manager is a growing trend:

- All WebSphere family products will be integrated with Tivoli Access Manager over time.
- Tivoli Access Manager will eventually support local operating system and custom registries in addition to the currently supported LDAP registries.
- These enhancements will provide a unified management infrastructure to handle authorizations across all WebSphere family products.

Security, networking, and topology considerations

Clearly, as the LDAP server contains and the Access Manager server manages sensitive data in terms of authentication, authorization and privacy, the servers belong in the data layer of the network. It is best practice to enable Secure Socket Layer configuration options between the databases so that the data is encrypted.

Legal considerations (privacy and data protection)

You should be aware that there might be some legal and or regulatory issues that surround storing of certain data types such as personally identifiable data in the European Union on IT systems. Ensure that you have consulted your legal department before deploying such information about your systems. These considerations vary by geography and industry, and it is beyond the scope of this book to discuss specific issues.

5.2 IBM Directory Server

This section looks at IBM Directory Server and its integration with WebSphere Application Server V6.

5.2.1 The Lightweight Directory Access Protocol

A directory is a data structure that allows the look up of names and associated attributes arranged in a hierarchical tree structure. In the context of enterprise application servers, this allows applications to look up a user principal and determine what attributes the user has and of which groups the user is a member. Decisions on authentication and authorization can then be made using this information.

LDAP is a fast and simple way of looking up user entities in a hierarchical data structure. It has advantages over simply using databases as a user repository in terms of speed, simplicity and standardized models or schemas for defining data. Standard schemas have standard hierarchies of objects such as objects which represent a person in an organization. These objects in turn have attributes such as a user ID, common name, and so forth. The schema can also have custom objects added to it which means that your directory is extensible and customizable.

Generally, LDAP is chosen over a custom database repository of users for these reasons. LDAP implementations (such as IBM Directory Server) uses database engines under the covers, but these engines are optimized for passive lookup performance (through indexing techniques). This is possible because LDAP implementations are based on the assumption that the data changes relatively infrequently and that the directory is primarily for looking up data rather than updating data.

Today, there are many LDAP server implementations. For example, IBM (Tivoli) Directory Server, iPlanet Directory Server, Open LDAP’s SLAPD server, and Microsoft Active Directory all support the LDAP protocol.

5.2.2 Supported LDAP servers for WebSphere Application Server V6

WebSphere Application Server V6 supports IBM (Tivoli) Directory Server, IBM Secureway Directory Server, Microsoft Active Directory, Domino, eDirectory, and custom LDAP directories. Other LDAP conformant servers might function, but this is not guaranteed. Other directories can also be configured by using the custom directory settings and filters.

5.2.3 IBM Directory Server and WebSphere Application Server V6

To use security features such as form-based login for Web applications that run in your application server and to secure a stand-alone application server or cell of application servers, Global Security settings in WebSphere Application Server need to be configured. Essentially, WebSphere Application Server must be told about the user registry to be used (in this case, an LDAP registry). Then, Global Security must be enabled. Global Security cannot be enabled without being told of the registry to be used first.

These tasks can be done via the WebSphere administrative console or via the wsadmin command line tool. For detailed instructions on how to set up these configurations, refer to *WebSphere Application Server V6 - Security Handbook*, SG24-6316.

For more information about the available alternatives for security in WebSphere Application Server V6, refer to Chapter 13, “Planning for system management” on page 371 and Chapter 17, “Planning for security” on page 487.

Additionally, refer to the WebSphere Application Server V6 information center:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

Security, networking, and topology considerations

Because the LDAP server contains sensitive data in terms of authentication, authorization and privacy, the LDAP server belongs in the data layer of the network. It is best practice to enable Secure Socket Layer options in the WebSphere Application Server V6 Global Security configuration so that the data is encrypted between the application server layer and the data layer.

Legal considerations (privacy and data protection)

There might be some legal and or regulatory issues that surround storing of certain data types such as personally identifiable data in the European Union on IT systems. Ensure that you have consulted your legal department before deploying such information about your systems. These considerations vary by geography and industry, and it is beyond the scope of this book to discuss specific issues. Legal considerations could become even more of an issue when you create custom objects and attributes in the LDAP directory schema that can store further information relating to individuals.

5.3 WebSphere Business Integration

Business integration refers to the integration of data, applications, processes, and people across an enterprise and across enterprises. WebSphere Business Integration is an IBM software brand that encompasses software offerings that support business integration.

To understand these offerings and how they integrate with WebSphere Application Server V6, first you need to understand two related concepts: Service Oriented Architectures and the Enterprise Service Bus.

5.3.1 Service Oriented Architectures

At the heart of the concept of Business Integration is the Service Oriented Architecture (SOA). An SOA is where legacy and new applications in an enterprise and across enterprises are seen as a set of services. Each service has a common interface that is based on open standards. Generally, this common interface standard is the set of open standards that define creation of a Web service or Java Messaging Service.

Enterprises often have applications and data sources on disparate platforms, communicating with a variety of protocols and implemented in various programming language. Implementing an SOA transforms these resources into a set of modular services that can interact uniformly and be reorganized and reintegrated quickly to react to changing business needs.

5.3.2 Enterprise Service Bus

A closely related concept to an SOA is the Enterprise Service Bus pattern. The Enterprise Service Bus is a middleware design pattern that attempts to integrate disparate applications, data sources, platforms languages, and platforms.This integration is done through the implementation of Web services and messaging infrastructure. This pattern should also support transformation and routing technology.

5.3.3 WebSphere Business Integration family of technologies

The WebSphere Business Integration brand encompasses a number of software technologies that offer various business integration capabilities. Some of the key software offerings are:

- WebSphere Business Integration Connect integrates business-to-business trading communities.

- WebSphere Business Integration Adapters integrate a wealthy of proprietary software such as SAP, Microsoft Exchange, and so forth.
- WebSphere Business Integration Message Broker builds, routes, and transforms WebSphere MQ messages.
- WebSphere Business Integration Event Broker provides publish and subscribe mechanisms within WebSphere Business Integration Message Broker.
- WebSphere Business Integration Server Foundation is an application server that is enhanced for business integration and that is built on top of WebSphere Application Server.
- WebSphere Business Integration Interchange Server and Toolset is a message integration broker that uses WebSphere Business Integration Adapters to allow heterogeneous applications to exchange data.

This is not an exhaustive list, such a list would be beyond the scope of this book. This list however does show the richness of the Business Integration offerings that are available. WebSphere MQ (the proprietary IBM messaging solution) is the asynchronous protocol that allows these solutions to communicate. These technologies can all form or communicate within an Enterprise Service Bus architecture. It is against this back drop that we now turn to examine WebSphere Application Server V6 and how it too can integrate with this middle-ware pattern.

5.3.4 WebSphere Application Server and Business Integration

The evolution of the WebSphere Application Server software family has been relatively complex regarding business integration technologies. The general trend over time however has been that:

- Each application server offering supports open standards such as support for Web services, JMS, and so forth, which all contribute to business integration as they are fundamental to SOA.
- The higher level versions of the application server (such as WebSphere Application Server Enterprise Edition) have supported more sophisticated business integration technologies such as support for BPEL4WS and work flow choreographer technologies.
- Over time, some of these more sophisticated business integration technologies are supported in lower versions of WebSphere Application Server.

WebSphere Business Integration Foundation Server

With the release of WebSphere Application Server 5.1 the Enterprise Edition version of the server was rebranded to WebSphere Business Integration. It was also renamed from *WebSphere Application Server Enterprise Edition* to *Foundation Server*. This reflected the product's strong relationship with other products in the WebSphere Business Integration family. WebSphere Business Integration Server Foundation is optimized for SOA integration.

Particular features of note are the support for BPEL4WS (Business Process Execution Language for Web Services) and process choreographer technologies. These features are in turn supported by the WebSphere Studio Application Developer Integration Edition IDE.

WebSphere Application Server V6

The two key interface technologies for a Enterprise Service Bus are messaging and Web services technologies. With the release of WebSphere Application Server V6 Java Messaging and Web services technologies are integrated into a SOA implementation called a service integration bus.

The service integration bus is the key concept in default messaging provider for WebSphere Application Server V6. It is part of the fully-compliant Java Messaging Service (JMS 1.1) specification which requires a pure Java Implementation of a messaging provider.

The service integration bus fulfills the Enterprise Service Buses requirement for both messaging and Web services interfaces to the Bus by exposing message queue end points as Web services through a WSDL document. It provides high availability through clustering and provides logging, data transformation and routing configuration through the use of mediations which are programmable filters that can be applied to the JMS queue and Web Service end points on the bus.

The service integration bus can be seen as a micro implementation of a Enterprise Service Bus. It allows WebSphere Application Server V6 servers and clusters of servers to exchange information via pure Java implementation of a Messaging Service and Web service interfaces. This WebSphere Application Server specific architecture can then integrate at a macro level with a wider Enterprise Service Bus whose communication is implemented in a proprietary messaging software technology (namely WebSphere MQ).

Figure 5-2 on page 166 gives a high level overview of the SOA possibilities with WebSphere Application Server V6 and the WebSphere Business Integration Family of products.

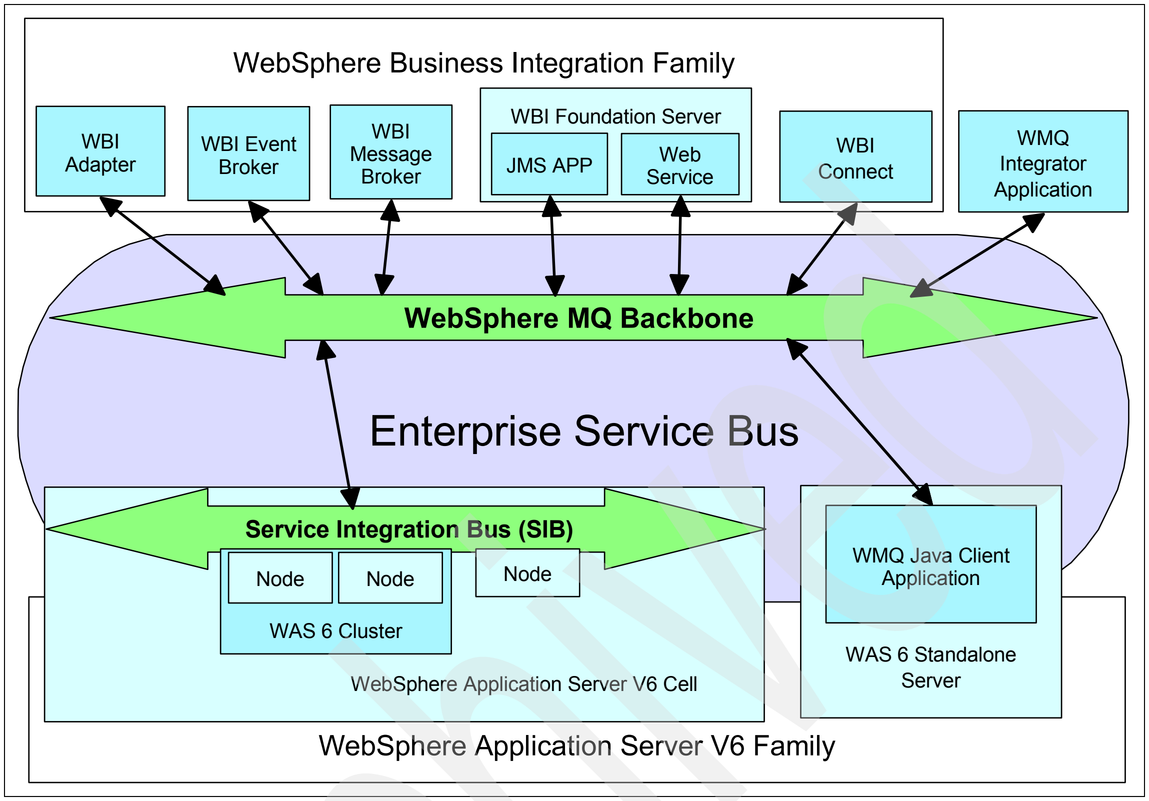

WebSphere Business Integration Family
WBI Adapter
WBI Event Broker
WBI Message Broker
WBI Foundation Server
JMS APP
Web Service
WBI Connect
WMQ Integrator Application
WebSphere MQ Backbone
Enterprise Service Bus
Service Integration Bus (SIB)
Node
Node
Node
WAS 6 Cluster
WebSphere Application Server V6 Cell
WMQ Java Client Application
WAS 6 Standalone Server
WebSphere Application Server V6 Family

Figure 5-2 SOA possibilities

The Enterprise Service Bus is comprised of a WebSphere MQ backbone and a WebSphere Application Server V6 service integration bus. Note that:

- WebSphere Application Server V6 nodes (and clusters) can communicate with one another via the service integration bus.
- WebSphere Application Server V6 applications are also able to communicate with the WebSphere MQ backbone via a link from the service integration bus or directly through (for example) a WebSphere MQ client for Java connection.
- A Web service application running in WebSphere Business Integration Foundation Server is also shown connecting to the backbone.

For more information about the WebSphere Messaging Platform that is implemented in WebSphere Application Server V6, see Chapter 15, “Planning for messaging” on page 433.

Summary

The WebSphere Business Integration family of products provides a rich combination of solutions for integrating disparate business resources across the enterprise and between enterprises. WebSphere MQ is the transport that underlies these technologies. The underlying concept or pattern is the Enterprise Service Bus. WebSphere Application Server V6 has its own micro implementation of an Enterprise Service Bus in the form of its service integration bus. This allows WebSphere Application Server V6 servers to communicate using Java Messaging Service and Web service technologies in a pure Java implementation. The service integration bus can link to a wider WebSphere MQ backbone. It is possible to use JMS and WebSphere MQ proprietary clients to connect to WebSphere MQ infrastructures with out using the service integration bus features.

5.4 MQ integration

WebSphere MQ is proprietary, asynchronous messaging technology that is available from IBM. WebSphere MQ is middle-ware technology that it is designed for application-to-application communication rather than application-to-user and user interface communication.

WebSphere MQ is available on a large number of platforms and operating systems. It offers a fast, robust, and scalable messaging solution which assures once, and once only, delivery of messages to queue destinations which are hosted by queue managers. WebSphere MQ is the underlying transport for much of Business Integration solutions from IBM. The messaging solution has APIs in C, Java, and COBOL which allow applications to construct, send, and receive messages.

With the advent of JMS, generic, portable client applications can be written to interface with proprietary messaging systems such as WebSphere MQ. The integration of WebSphere Application Server V6 with WebSphere MQ over time has been influenced by this dichotomy of generic JMS and proprietary WebSphere MQ access approaches.

5.4.1 Supported JMS messaging providers

With the release of WebSphere Application Server V6 the messaging platform has been changed considerably. In the previous version, the JMS providers that could be configured were:

- An embedded messaging provider (that was WebSphere MQ under the covers).
- A separately installed WebSphere MQ Server installation.
- An option to configure a generic JMS provider implementation.

In this new release, the JMS provider options have the following features:

- The embedded WebSphere MQ JMS messaging provider is removed and replaced with the default, platform messaging provider.
- A separate WebSphere MQ 5.4 server installation can still be configured as the JMS messaging provider.
- Generic JMS providers can still be configured as before.

Default Messaging Provider (default messaging provider)

This removal of the embedded WebSphere MQ JMS provider is a response to the JMS specification which requires the JMS provider to be a pure Java application. It is also motivated by an effort to further integrate security and clustering for the messaging provider into the base WebSphere Application Server.

The default messaging provider's resources in the application server are maintained and managed by a process called messaging engine. Messaging engines maintain configuration and state information in database tables. Messaging engines provide the connection point when applications put and get messages to queue destinations.

Because this default messaging provider supports the concept of a service integration bus (which in turn supports both JMS and Web service interfaces), the implementation of this provider is part of a general IBM approach to support SOA and the concept of the Enterprise Service Bus middleware design pattern.

5.4.2 Integration scenarios and possibilities

The changes to messaging providers in WebSphere Application Server V6 allow for backwards compatibility with messaging clients and architectures.

Client interoperability

WebSphere Application Server V6 allows compatibility with previous releases. The following list highlights client interoperability between V5 and V6:

- Existing WebSphere Application Server V5 embedded JMS clients can connect to WebSphere Application Server V6 queue destinations.
- WebSphere Application Server V6 applications can connect to WebSphere Application Server V5 embedded messaging providers.
- WebSphere Application Server V5 embedded messaging client is still installed in WebSphere Application Server V6 to allow mixed environments and migrations.
- However WebSphere Application Server V5 embedded messaging providers cannot connect to a WebSphere Application Server V6 service integration bus.
- An MQ client link can be created to support any legacy MQ clients to talk to the WebSphere Application Server V6 messaging engines.

Migration of V5 Embedded Messaging Provider to V6

During an upgrade to WebSphere Application Server V6 the following changes are made automatically:

- When upgrading a single server to WebSphere Application Server V6, the existing embedded JMS server is replaced with a messaging engine.
- When upgrading an existing WebSphere Application Server Network Deployment V5 node, the embedded JMS server is converted to an application server hosting a messaging engine.
- Any existing JNDI resources that is relied upon (such as QueueConnectionFactories) remain unchanged. A WebSphere Application Server V5 emulation mode runs and handles the mappings of legacy JNDI resources to the new default messaging provider resources. These legacy applications settings can be migrated fully, by hand, over time if required.

WebSphere MQ interoperability

WebSphere MQ and WebSphere Application Server V6 Platform Messaging are complementary technologies that are tightly integrated to provide for various messaging topologies.

Coexistence of messaging engines and WebSphere MQ

WebSphere MQ and related business integration products such as WebSphere MQ Integrator and Event Broker are separate products to the default default messaging provider in WebSphere Application Server V6. WebSphere MQ can coexist on the same host as a WebSphere Application Server V6 messaging engine. Their installs binaries are completely separate.

MQ Link

Connectivity between an messaging engine and a WebSphere MQ queue manager is achieved by defining an MQ Link. When the MQ Link is created and used, WebSphere MQ sees the connected messaging engine as just another WebSphere MQ queue manager. This link converts between formats and protocols used by WebSphere MQ and the generic default messaging provider. WebSphere MQ Applications can now send messages to WebSphere Application Server V6 queue destinations and WebSphere Application Server V6 messaging applications can now send messages to WebSphere MQ queues. The messaging engine that has the MQ Link defined against it is known as a Link Engine. This Link Engine must not be part of a cluster. A service integration bus can have multiple Link Engines but each of these must connect to a different WebSphere MQ queue manager.

An MQ Link is configured via the WebSphere administrative console. Information such as the name of the queue manager and the name of the sender and receiver channels as defined under the remote WebSphere MQ queue manager are required for this configuration.

Note that:

- WebSphere MQ to service integration bus connections are only supported over TCP/IP.
- A messaging engine cannot participate in a WebSphere MQ cluster.

Recommended usage scenarios

Whether or not to use pure Java default messaging provider provider or to use the direct WebSphere MQ messaging provider depends on a number of factors. The basic, best practice guide is to use a default messaging provider if:

- You are currently using WebSphere Application Server V5 embedded messaging provider for intra-WebSphere Application Server messaging.
- You require messaging between WebSphere Application Server and an existing WebSphere MQ backbone and its applications.

Use WebSphere MQ Messaging Provider if:

- You are currently using a WebSphere MQ Messaging provider and simply wish to continue using it.
- You require access to heterogeneous, non JMS applications.
- You require access to WebSphere MQ clustering.
- You require access to WebSphere MQ functions that are not supported by JMS.

For more information about the default messaging provider that is implemented in WebSphere Application Server V6, see Chapter 15, “Planning for messaging” on page 433.

For more information about WebSphere MQ, see:

http://www-306.ibm.com/software/integration/wmq/

6

Development tools

This chapter gives an overview of the development tools that are available for the WebSphere Application Server V6 environment. It contians the following sections:

- Development and deployment tools
- Performance and analysis tools
- End-to-end application life cycle

6.1 Development and deployment tools

The WebSphere Application Server V6 environment comes with a rich set of development tools. WebSphere Application Server - Express V6 comes with the Application Server Toolkit and Rational Web Developer. WebSphere Application Server Network Deployment V6 comes with the Application Server Toolkit and a trial version of Rational Application Developer. For a full version of Rational Application Developer, additional licensing is required.

This section covers the development and deployment tools that are available and their key features.

6.1.1 Rational tools

Rational development tools specifically aim at the development and deployment of applications to the WebSphere Application Server V6 environment. The tool set runs within the Eclipse 3.0 Integrated Development Environment (IDE) framework. The Eclipse framework is widely used and is free for IDE development. Plug-ins can be developed for the Eclipse framework. So, in addition to the Rational plug-ins and the eclipse base, which make up Rational Application Developer, there are also plug-ins that are available from other vendors that you can use with this same Eclipse framework. IBM is committed to supporting and using such open standards.

In short, Rational development tools are optimized for WebSphere Application Server V6 development. At the same time, these tools are generic and highly extensible for any Web or Enterprise Java development project.

Rational Application Developer comes with many standard Eclipse plug-ins as well as specific Rational plug-ins. For more information about Eclipse, see:

http://www.eclipse.org

For information about plug-ins, see:

http://www.eclipseplugincentral.com

In addition, Eclipse and, thus, Rational Application Developer, come with a plug-in development toolkit so that you can develop your own plug-ins if necessary.

Rational Application Developer is the super set of all the Rational IDE applications. Figure 6-1 on page 175 gives an overview of the IDE software sets of functionality and lists the features in each of these sets.

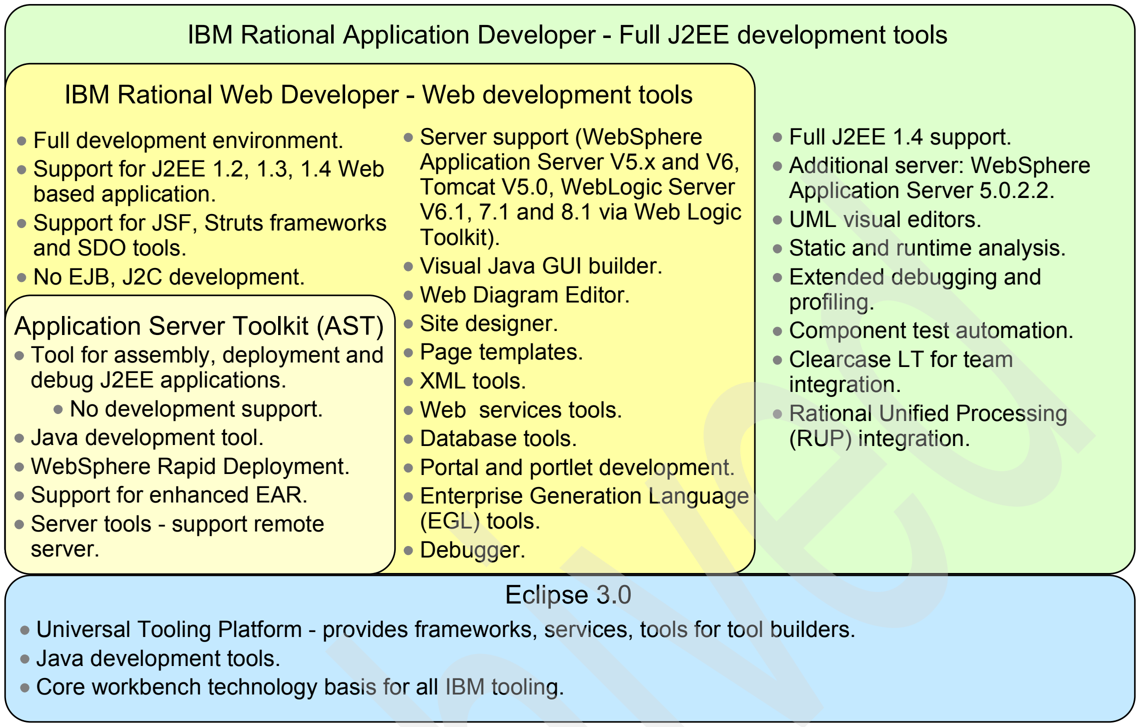

IBM Rational Application Developer - Full J2EE development tools
IBM Rational Web Developer - Web development tools
Full development environment.
Support for J2EE 1.2, 1.3, 1.4 Web based application.
Support for JSF, Struts frameworks and SDO tools.
No EJB, J2C development.
Application Server Toolkit (AST)
Tool for assembly, deployment and debug J2EE applications.
No development support.
Java development tool.
WebSphere Rapid Deployment.
Support for enhanced EAR.
Server tools - support remote server.
Server support (WebSphere Application Server V5.x and V6, Tomcat V5.0, WebLogic Server V6.1, 7.1 and 8.1 via Web Logic Toolkit).
Visual Java GUI builder.
Web Diagram Editor.
Site designer.
Page templates.
XML tools.
Web services tools.
Database tools.
Portal and portlet development.
Enterprise Generation Language (EGL) tools.
Debugger.
Full J2EE 1.4 support.
Additional server: WebSphere Application Server 5.0.2.2.
UML visual editors.
Static and runtime analysis.
Extended debugging and profiling.
Component test automation.
Clearcase LT for team integration.
Rational Unified Processing (RUP) integration.
Eclipse 3.0
Universal Tooling Platform - provides frameworks, services, tools for tool builders.
Java development tools.
Core workbench technology basis for all IBM tooling.

Figure 6-1 A summary of the features in Rational Application Developer

6.1.2 Rational Web Developer

Rational Web Developer is the successor to WebSphere Studio Web Developer. It is a subset of the functionality in Rational Application Developer. This sub set of functions focuses on Web development tooling. JavaServer Pages, Servlets, Struts, JavaServer Faces, static HTML, XML, and Web services development are all supported. However, if you require development of full J2EE applications, including EJB development, then you need Rational Application Developer.

The following sections discuss the key features of Rational Web Developer.

Latest Web Services support

The Web IDE allows you to build and consume Web services. Support is included for Java Bean and EJB Web services as per the Java Specification Requests JSR-101 and JSR-109. There is also support for setting a Web service conformance level to basic profile 1.0, simple SOAP basic profile 1.0, and attachments profile 1.0.

Wizards are available for generating RMI stubs and skeletons for JAX-RPC handlers. A SOAP Monitor is integrated into the Web service creation wizard to allow the viewing of traffic in Web services. Secure Web service support is provided by wizards to update the Web service deployment descriptors with Web Service Security settings.

Rapid Web Development

The Web IDE includes support for pages templates to give a consistent look and feel to Web sites. The Web site designer functionality allows you to nest page templates and to develop navigation bars. There are also visual editors for the rapid creation of Struts applications.

JavaServer Faces

The Web IDE now supports JavaServer Faces (JSF) 1.0. There are also additional IBM JSF components that are included as well as visual diagrams and editors to allow the clear layout of actions and JSF page navigation. In addition, new client components add new formats for displaying data.

Service Data Objects

Service Data Objects (SDOs) offer a generic API for accessing a large number of datasources. Mediator interfaces are used to access disparate datasource formats. Data is presented in JSF front ends, allowing the Web designer to focus on data presentation without the concern of a large number of datasource access APIs. The access to the different datasources is transparent to the developer.

Portal application development

Portals are Web sites that provide a single point of access to applications and information. Rational Web Developer provides Struts and JSF support for building portal applications. Struts is a framework for providing Model View Controller architecture for a Web application, while JSF is a framework for providing “off-the-peg” presentation components in a Web application. The IDE comes with SDOs for accessing and updating Siebel and SAP records, which can be added to portal applications.

Enterprise Generation Language support

Enterprise Generation Language (EGL) is high-level language that is used for developing business logic. This language is implementation independent, and developers who are unfamiliar with specific languages, such as Java and COBOL, can use EGL. Implementation-specific code in Java or COBOL is then generated form the EGL. Rational Web Developer has wizards and tools that support EGL. For more information about EGL, see:

http://www.ibm.com/developerworks/websphere/zones/studio/egldocs.html

6.1.3 Rational Application Developer

Rational Application Developer includes all the subset features of Rational Web Developer. It is the IDE of choice for developing and deploying applications for WebSphere Application Server V6. It is the successor to WebSphere Studio Application Developer and supports applications that are developed for WebSphere Application Server versions 4.0, 5.0 and 6.0.

The following sections discuss the key features of Rational Application Developer.

Full support provided for J2EE 1.4

All wizards have J2EE 1.4 options. Deployment descriptor editors have been upgraded for J2EE 1.4. options. Wizards are provided for the migration of J2EE 1.2 and 1.3 applications to J2EE1.4, and there is full support of the latest enterprise Java APIs (Servlet 2.4 , JSP 2.0, and EJB 2.1).

Annotation-based programming

New EJB, Servlet, and Web services annotations allow programmers to use xDoclet style tags in source code files to specify configuration parameters for J2EE applications. Thus, the artifacts on which the programmer has to concentrate are reduced. For further details, see 12.3, “Annotation-based programming” on page 349.

Modelling functionality

The IDE has Unified Modeling Language (UML) functionality, which allows the developer to express classes, EJBs, and database schemas visually. New UML Topic and sequence diagrams can be used to view class relationships and method interactions. Because the diagrams are created by wizards and by dropping classes on diagram panes, only a passive knowledge of UML is required on the part of the developer. That is, the developer only needs to be able to read the diagrams rather than actively having to know the language to generated them.

Integration with the Rational Unified Process and Tool Set

Rational also offers an extensive suite of tools for the end-to-end application development life cycle. These tools are not included with Rational Application Developer. Instead, they are additional tools that can interact with the IDE.

For example, the Rational Tools Suite includes Rational ClearCase (a code repository solution) and Rational RequisitePro® (a requirements analysis tool) in addition to many others. Rational Application Developer integrates seamlessly with these additional tools and also with the Rational Unified Process which is the iterative software development process that Rational software supports. The

IDE has a Process Browser and Process Advisor view which allows the developer to view Rational Unified Process best practice advice for the task in hand. For further information see, 11.3, "End-to-end life cycle" on page 323.

Application analysis

The IDE also has built -n wizards for analyzing applications for coding practices. Inefficient coding practices are identified in applications, and examples are provided for best practices and issue resolution.

Unit testing

Rational Application Developer provides features for the easy creation, execution, and maintenance of unit tests for J2EE components. It supports creating unit tests for:

- Java classes
- EJBs (1.1, 2.0 and 2.1)
- Web services (J2EE and .Net based)

A selection of test patterns are available for defining complex unit test cases. Test data can be stored in a separate data pool for the flexible definition of test cases.

The unit test framework in Rational Application Developer is constructed using JUnit and Haydes technology. JUnit is a framework for unit testing in Java. It is available as a separate command line tool, and it is also available as an Eclipse plug-in. Haydes is an Eclipse sub-project used to produce an integrated test, trace, and monitoring environment. Recently, Hyades has been incorporated into a larger Eclipse project for test and performance tools. This is an example of how Rational Application Developer via the Eclipse IDE framework is leveraging wider open standards technologies.

For further information about using JUnit, refer to Chapter 11, "Planning for application development" on page 305. Additionally, see the JUnit and Hyades sites at:

http://www.junit.org
http://www.eclipse.org/hyades

6.1.4 Application Server Toolkit

The Application Server Toolkit is a suite of tools for deploying applications to application servers. The Application Server Toolkit is a sub set of functionality in Rational Web Developer functionality and, thus, also a sub set of Rational Application Developer functionality.

Features in the Application Server Toolkit include tools for the following operations of J2EE applications:

- Assembling
- Deploying
- Running
- Debugging
- Profiling

Applications can be assembled and then deployed to local or remote test servers. When deployed, you can run the applications, debug them to diagnose errors, or profiled them to analyze performance. For more information about application deployment, see: Chapter 12, “Planning for application deployment” on page 345.

6.1.5 WebSphere Rapid Deployment

One of the new features in WebSphere Application Server V6 is the WebSphere Rapid Deployment. This feature allows for the deployment of applications with minimum effort on the part of the developer or administrator. The Rapid Deployment model has three basic features which allow for ease of deployment:

- Annotation-based programming
- Deployment automation
- Enhanced EAR file

Annotation-based programming allows you to annotated the EJB, Servlet, or Web service module code with special JavaDoc syntax annotations. When the source of the module is saved, the directives in these annotations are parsed, and the IDE uses the directives to update deployment descriptors. Thus, the developer can concentrate on the Java source code rather than metadata files.

Deployment automation is where applications that install packages are dropped into a hot directory under an application server and the application is installed automatically. Any installation parameters that have not been specified by the install package’s deployment descriptors have default values that are applied by the automated deployment process.

Rapid deployment is more sophisticated and easy to use, because it allows for a free-form deployment operation. In this mode, it is possible to drop deployment items into the hot directory without strict J2EE packaging such as Web or Enterprise archives and deployment descriptors. It is possible, for example, for annotations in Java source code to be processed at deployment time. This information is then used to build deployment descriptors.

An enhanced EAR file allows the Enterprise archive package to include information about resources and properties, such as datasources. Thus, you can specify more information about the application and how it should be deployed, which means you have fewer decisions to make at deployment time.

6.1.6 Integration with ClearCase

Rational Application Developer has a client for accessing Rational ClearCase LT. Rational ClearCase LT is the junior member of the Rational ClearCase family of Source Code Management (SCM) repositories. Rational Application Developer also has a client for accessing CVS (Concurrent Versions System), which is another popular SCM. The Rational ClearCase LT client is an example of how the IDE integrates with the wider suite of Rational tools and the Rational Unified Process. For more information about Rational ClearCase, see 11.1, “Versioning” on page 306.

6.2 Performance and analysis tools

Profiling tools are available in Rational Application Developer to identify and analyze various code performance issues in an application. These performance issues generally fall into four major categories:

- Memory leaks
- Performance bottlenecks
- Excessive object creation
- System resource limits

The information required to diagnose these issues is gathered while the application is running. The application can be running in an application server, or the application can be a stand-alone Java application. The application can be profiled locally or on a remote machine.

In order to profile applications on a remote machine, an Agent Controller application must be installed and running on the remote machine. The Rational IDE connects to the Agent Controller, which then runs Profiler Agents, which in turn control and analyze a specific Java Virtual Machine (JVM) in which the application is running. The JVM can be in WebSphere Application Server (if so the application server must be started in profiling mode), or it can be a JVM running a simple stand-alone Java application.

Profiling Sets can be configured. Thus, within Rational Application Developer profiling GUIs, the developer can chose which sets of profiling types to analyze. For example, the developer might focus on memory leak analysis rather than on code execution coverage. The profiling sets and the profiling types that they contain allow for the fine-grained control of what type of analysis is to be run.

The following sections discuss the profiling sets that are available.

6.2.1 Memory analysis

The memory analysis can consist of:

- Memory usage
- Memory leak analysis manual
- Memory leak analysis automatic

The manual leak analysis allows more fine-grained control, where as the automatic analysis selects default settings. The leak analysis tooling profiles the application and then saves memory heap dumps of the profile. You can select the heap dump to analyze. The tooling then identifies candidate objects that are likely candidates as the source of memory leaks. The tooling also displays the allocation path for the leak candidate so that you can see where the memory is allocated in the path of execution of application.

6.2.2 Thread analysis

Thread analysis is done to examine contention of execution between threads and possible thread dead lock. In thread analysis, there are three profile sets:

- Thread view

  The thread view is a graphical representation of the threads in the application and the state of those threads. Information about locks and which threads are holding those locks is indicated.

- UML2 sequence diagram view

  The UML2 sequence diagram view indicates the flow of method calls and is synchronized with the thread view.

- Profiling monitor

  The profiling monitor shows the call stack for each executing thread.

6.2.3 Execution time analysis

Execution time analysis is used to detect performance bottlenecks, that is which parts of code in the execution are taking the longest time to execute. Various graphical and tabular views are available to examine the execution history of the application. Representations of the entire application execution or detailed, method-level analyses are available. For example, a global execution diagram represents the flow of execution between objects. Selecting a particular object opens a method view to show method execution within that object.

6.2.4 Code coverage

Code coverage is the analysis of how much each part of code in an application is executed (if at all). You can use code coverage to identify redundant pieces of code which can be removed or refactored. The code coverage profile set includes analysis of code execution at the method level and down to specific lines of code. The views that are available of this information are graphical representations of classes, methods, and lines of code. Tabular and graphical representations of statistical data for code coverage are also available.

6.2.5 Probe kit

The probe kit is a new feature that you can use to insert Java code fragments into an application during execution. The probe kit uses the byte-code instrumentation (BCI) framework. The probes (code fragments) are saved in a separate project and can be reused.

Rational Application Developer provides probe editors so that you can write probes easily. When the application is profiled (executed against a profile set), the probe profile set is chosen. Specific probes are chosen depending on what the profiling needs to achieve.

Probes can be inserted at:

- Method call locations
- Method call entry and exit
- In a catch or finally block
- Before original code in a class static initializer

Probes can access:

- Package class and method names
- Method signatures
- A `this` object
- Arguments and return values
- Exceptions

For example, you might write a probe that is inserted at the start and end of every method. The probe would access the class and method name and print them out to the standard output stream. This probe would then be a generic probe for tracing an application to see which parts of the code are being executed. You could reuse this probe with any application, which would save the time consuming insertion of tracing statements in an application and which can be misleading if an entry or exit statement (or both) is omitted by mistake. Clearly, there are many possibilities for the use of the probe kit. Development teams can develop a library of probes in a separate Eclipse project, which would then be used for profiling specific Java development projects.

6.3 End-to-end application life cycle

IBM Rational and WebSphere software offer tools for the end-to-end application life cycle. The following sections provide an overview of the tools and where these tools fit in the application life cycle. The application life cycle divides broadly into six stages as illustrated in Figure 6-2.

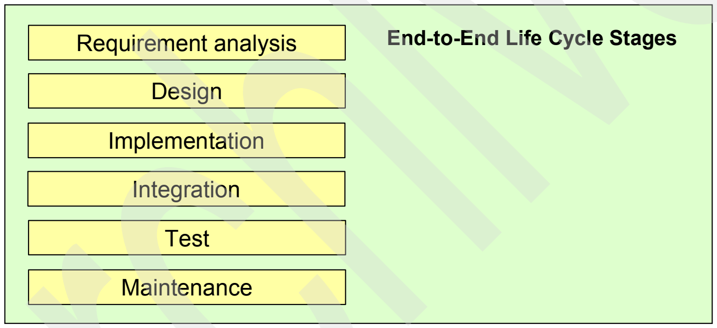


Figure 6-2 Stages of the application life cycle

6.3.1 The place for Rational Application Developer in the life cycle

Rational Application Developer and its subset tools (Rational Web Developer and Application Server Toolkit) have functionality that largely maps to the Implementation stage of the life cycle, with some functionality for the Integration and Test stages. As stated previously, Rational Application Developer has code generation wizards, unit testing tools, performance analysis tools, and tools for deploying applications. These IDE tools are optimized for use when building applications for deployment to the WebSphere Application Server environment. Figure 6-3 shows this relationship.

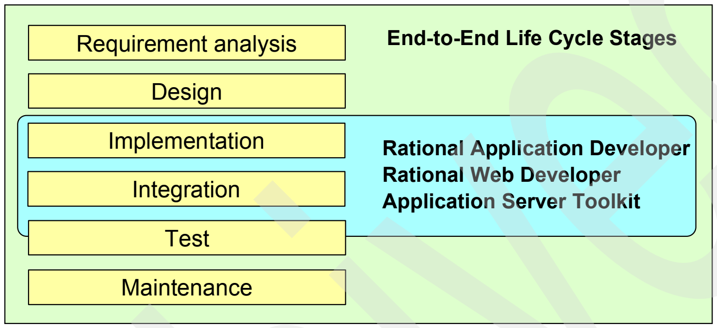


Figure 6-3 Rational Application Developer and its subset features in the life cycle

Additionally, because the Rational Application Developer has a client for Rational ClearCase LT and a Rational Unified Process browser, the IDE integrates with the broader Rational Software offerings.

6.3.2 The wider rational tool set and development life cycle

As depicted in Figure 6-4, the key stages of the life cycle process map to the key disciplines of requirement and analysis, design and construction, and software quality. The IBM Rational tools are shown grouped within these disciplines.

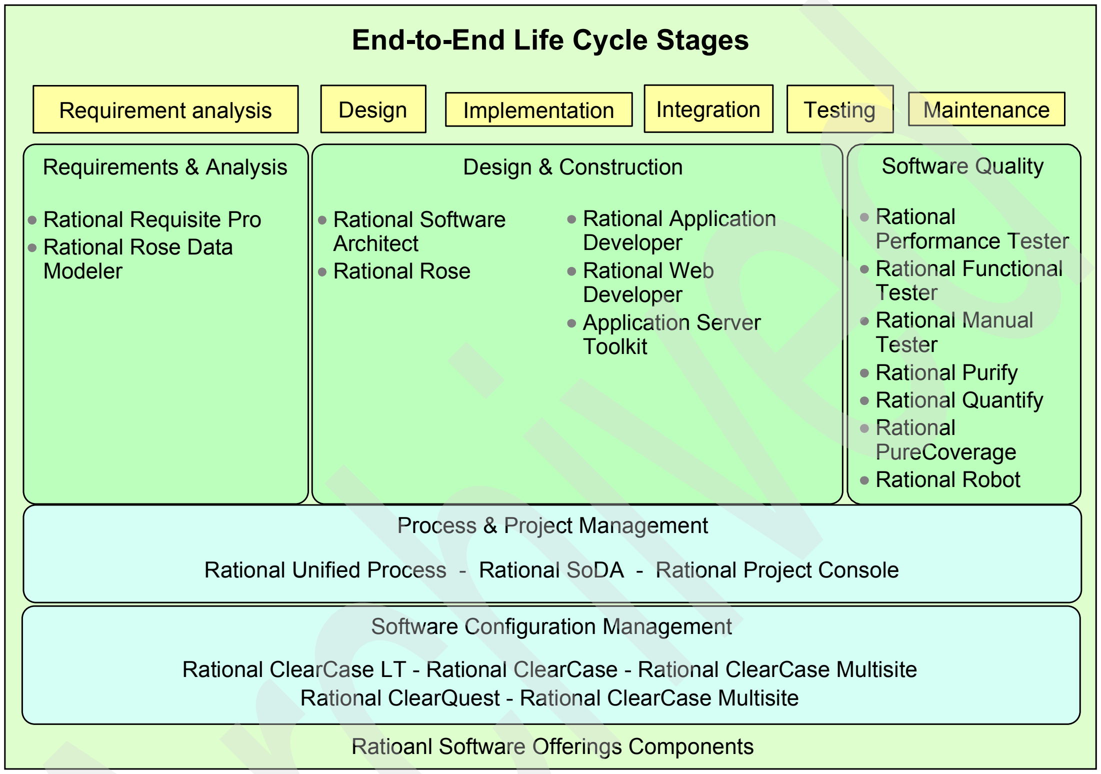

End-to-End Life Cycle Stages
Requirement analysis
Design
Implementation
Integration
Testing
Maintenance
Requirements & Analysis
Rational Requisite Pro
Rational Rose Data Modeler
Design & Construction
Rational Software Architect
Rational Rose
Rational Application Developer
Rational Web Developer
Application Server Toolkit
Software Quality
Rational Performance Tester
Rational Functional Tester
Rational Manual Tester
Rational Purify
Rational Quantify
Rational PureCoverage
Rational Robot
Process & Project Management
Rational Unified Process - Rational SoDA - Rational Project Console
Software Configuration Management
Rational ClearCase LT - Rational ClearCase - Rational ClearCase Multisite
Rational ClearQuest - Rational ClearCase Multisite
Ratioanl Software Offerings Components

Figure 6-4 Overview of Rational products in relation to other Rational components

Shown in the figure is a mapping of the end-to-end life cycle stages to discipline areas and then where the Rational tool components fit within these disciplines.

Rational ClearCase and Rational ClearQuest® together provide a comprehensive Software Configuration Management (including Source Code Management) and Unified Change Management infrastructure.

Rational RequisitePro offers tools for requirements analysis. Rational SoDA® is for automated documentation generation. Rational Rose is a tool for component design. Rational Purify®, PureCoverage®, and Quantify® are tools for analyzing performance of code to find bottlenecks and unused code as well as to generate data on code performance.

You can also group these component tools in terms of IBM offerings. Offerings packages are combinations of these components and specific licensing models which best suit particular team requirements. For example, packages that contain various combinations of these tools are available for smaller teams, larger teams, teams of developers, or teams that include developers, project managers, and testers. This variety of offering combinations allows you to select the best combination of tooling and the best licensing models for your purposes. Any particular component, such as Rational Robot for example, can or cannot be part of the offerings package, depending on the intended group of users.

For more information about Rational offerings, see:

`http://www.ibm.com/software/rational`

Part 2

Planning

This part of the book details the most common topologies as well as the topology selection criteria. It provides guidelines and best practices to plan for common activities such as:

- Installation
- Migration
- Application development and deployment
- Performance and high availability
- Security
- Messaging
- Web services
- System management and infrastructure

7

Planning for infrastructure

Wondering about how to plan and design an infrastructure deployment that is based on WebSphere middleware? This chapter covers the WebSphere-specific components that you have to understand in order to run a successful WebSphere infrastructure project. It contains the following sections:

- Infrastructure deployment planning
- Design for scalability
- Sizing
- Benchmarking
- Performance tuning

7.1 Infrastructure deployment planning

This section gives a general overview of which phases you have to go through during a project, how you gather requirements, and how to apply those requirements to a WebSphere project.

Typically, a new project starts with only a concept. Very little is known about specific implementation details, especially as they relate to the infrastructure. Hopefully, your development team and infrastructure team work closely together to bring scope to the needs of the overall application environment.

Bringing together a large team of people can create an environment that helps hone the environment requirements. If unfocused, however, a large team can be prone to wander aimlessly and to create more confusion than resolving issues. For this reason, you should consider carefully the size of the requirements team and try to keep the meetings as focused as possible. Provide template documents to be completed by the developers, the application business owners, and the user experience team. Try to gather information that falls into the following categories:

- Functional requirements, which are usually determined by the business-use of the application and are related to function.
- Non-functional requirements, which start to describe the properties of the underlying architecture and infrastructure like reliability, availability, or security.
- Capacity requirements, which includes traffic estimates, traffic patterns, and expected audience size.

Requirements gathering is an iterative process. There is no way, especially in the case of Web-based applications, to have absolutes for every item. The best you can do is create an environment that serves your best estimates, and then monitor the environment closely to adjust as necessary after launch. Make sure that all your plans are flexible enough to deal with future changes in requirements, and always keep in mind that the plans can impact other parts of the project.

With this list of requirements you can start to create the first drafts of your designs. You should target developing at least the following designs:

- Application design

  To create your application design, use your functional and non-functional requirements to create guidelines for your application developers about how your application is built.

This book does not attempt to cover the specifics of a software development cycle. There are multiple methodologies for application design, and volumes dedicated to best practices.

- Implementation design

  This design defines the target deployment infrastructure on which your application will be deployed.

  The final version of this implementation design will contain every detail about hardware, processors, and software that is installed on the different components. However, you do not begin with all these details. Initially, your implementation design simply lists component requirements, such as a database, a set of application servers, a set of Web servers, and whatever other components are defined in the requirements phase.

  This design might need to be extended during your project, whenever a requirement for change occurs or when you get new sizing information. Too often, however, the reality is that a project can require new hardware and, therefore, might be constrained by capital acquisition requirements. Try to not go back too often for additional resources after the project has been accepted.

  For more information about how to create the implementation design, see 7.2, “Design for scalability” on page 192.

With the two draft designs in hand, you can begin the process of formulating counts of servers, network requirements, and the other items related to the infrastructure. This exercise in sizing is described in 7.3, “Sizing” on page 208.

In some cases, it might be appropriate to do benchmark tests. There are many ways to perform benchmarking tests, and some of these methods are described in 7.4, “Benchmarking” on page 210.

The last step in every deployment is to tune your system and measure whether it can handle the projected load that your non-functional requirements specify. For more detail about how to plan for load tests, see 7.5, “Performance tuning” on page 211.

7.2 Design for scalability

Understanding the scalability of the components in your e-business infrastructure and applying appropriate scaling techniques can greatly improve availability and performance. Scaling techniques are especially useful in multi-tier architectures, when you want to evaluate components that are associated with IP load balancers, such as dispatchers or edge servers, Web presentation servers, Web application servers, data servers, and transaction servers.

7.2.1 Scaling your infrastructure

You can use the high-level steps described in this section to classify your Web site and to identify scaling techniques that are applicable to your environment: You should adapt this systematic approach to your situation.

7.2.2 Understanding the application environment

For existing environments, the first step is to identify all components and understand how they relate to each other. The most important task is to understand the requirements and the flow of the existing application(s) and what you can or cannot alter. The application is a significant contributor to the scalability of any infrastructure, so a detailed understanding is crucial to scale effectively. At a minimum, application analysis must include a breakdown of transaction types and volumes as well as a graphical view of the components in each tier.

Figure 7-1 on page 193 can aid in determining where to focus scalability planning and tuning efforts. The figure shows the greatest latency for representative customers in three workload patterns and on which tier the effort should be concentrated.

For example, for online banking, most of the latency typically occurs in the database server, while the application server typically experiences the greatest latency for online shopping and trading sites. Planning to resolve those latencies from an infrastructure solution becomes very different as the latency shifts.

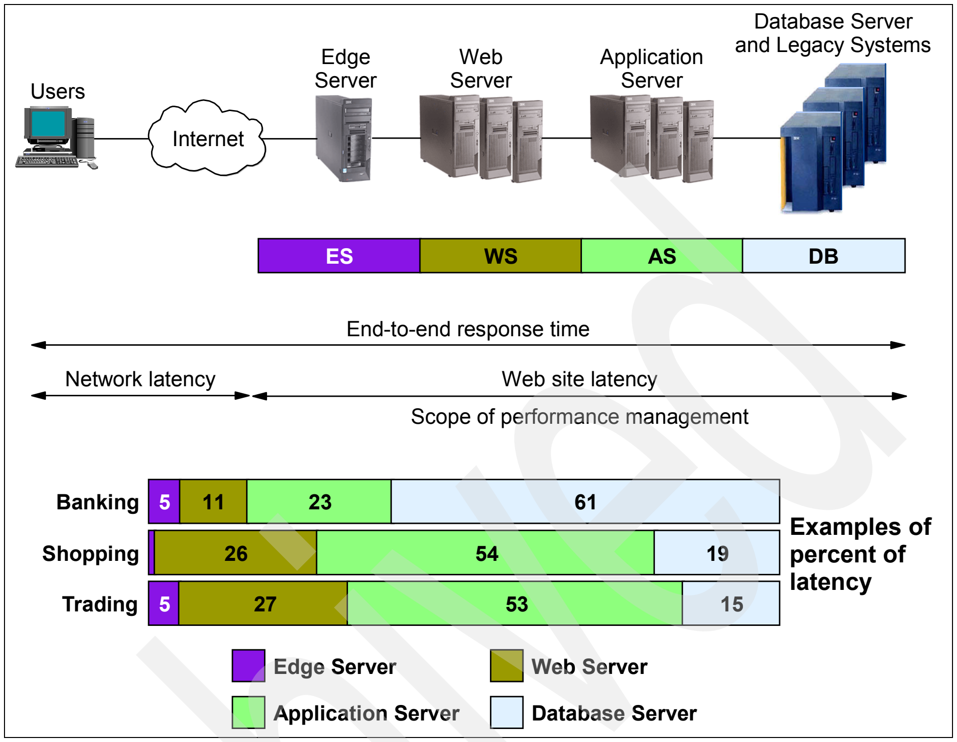


Figure 7-1 How latency varies based on workload pattern and tier

The way applications manage traffic between tiers significantly affects the distribution of latencies between the tiers, which suggests that careful analysis of application architectures is an important part of this step and could lead to reduced resource utilization and faster response times. Collect metrics for each tier, and make user-behavior predictions for each change implemented.

As requirements are analyzed for a new application, strive to build scaling techniques into the infrastructure. Implementation of new applications offer the opportunity to consider each component type, such as open interfaces and new devices, and the potential to achieve unprecedented transaction rates. Each scaling technique affects application design. Similarly, application design impacts the effectiveness of the scaling technique. To achieve proper scaling, application design must consider potential architectural scaling effects. You should follow an iterative, incremental approach in the absence of known workload patterns.

7.2.3 Categorizing your workload

Knowing the workload pattern for a site determines where you should focus scalability efforts and which scaling techniques you need to apply. For example, a customer self-service site, such as an online bank, needs to focus on transaction performance and the scalability of databases that contain customer information that is used across sessions. These considerations would not typically be significant for a publish/subscribe site, where a user signs up for data to be sent to them, usually via a mail message.

Workload patterns and Web site classifications

Web sites with similar workload patterns can be classified into site types. Consider the following distinct workload patterns and corresponding Web site classifications:

- Publish/subscribe (user to data)
- Online shopping (user to online buying)
- Customer self-service (user to business)
- Online trading (user to business)
- Business to business

Publish/subscribe (user to data)

Sample publish/subscribe sites include search engines, media sites such as newspapers and magazines, as well as event sites such as sports championships. Site content for publish/subscribe sites changes frequently, which drives changes to page layouts. While search traffic is low in volume, the number of unique items sought is high, resulting in the largest number of page views of all site types. Security considerations are minor compared to other site types. Data volatility is low. This site type processes the fewest transactions and has little or no connection to legacy systems. Table 7-1 on page 195 provides details about these workload patterns.

Table 7-1 Publish/subscribe site workload pattern

Workload pattern	**Publish/subscribe**
Categories/examples	► Search engines ► Media ► Events
Content	► Dynamic change of the layout of a page, based on changes in content, or need ► Many page authors; page layout changes frequently ► High volume, non-user-specific access ► Fairly static information sources
Security	Low
Percent secure pages	Low
Cross-session information	No
Searches	► Structured by category ► Totally dynamic ► Low volume
Unique Items	High
Data volatility	Low
Volume of transactions	Low
Legacy integration/complexity	Low
Page views	High to very high

Online shopping (user to online buying)

Sample online shopping sites include typical retail sites where users buy books, clothes, or even cars. Site content can be relatively static, such as a parts catalog, or dynamic, where items are frequently added and deleted, such as for promotions and special discounts that come and go. On a online shopping site, search traffic is heavier than on a publish/subscribe site, although the number of unique items sought is not as large. Data volatility is low. Transaction traffic is moderate to high and almost always grows.

When users buy, security requirements become significant and include privacy, non-repudiation, integrity, authentication, and regulations. Shopping sites have more connections to legacy systems, such as fulfillment systems, than publish/subscribe sites, but generally less than the other site types. Table 7-2 shows the online shopping site workload pattern.

Table 7-2 Online shopping site workload pattern

Workload pattern	**Online shopping**
Categories/examples	► Exact inventory ► Inexact inventory
Content	► Catalog either flat (parts catalog) or dynamic (items change frequently, near real time) ► Few page authors and page layout changes less frequently ► User-specific information: user profiles with data mining
Security	► Privacy ► Non-repudiation ► Integrity ► Authentication ► Regulations
Percent secure pages	Medium
Cross-session information	High
Searches	► Structured by category ► Totally dynamic ► High volume
Unique items	Low to medium
Data volatility	Low
Volume of transactions	Moderate to high
Legacy integration/complexity	Medium
Page views	Moderate to high

Customer self-service (user to business)

Sample customer self-service sites include those that are used for home banking, tracking packages, and making travel arrangements. Home banking customers typically review their balances, transfer funds, and pay bills. Data comes largely from legacy applications and often comes from multiple sources, thereby exposing data consistency.

Security considerations are significant for home banking and purchasing travel services, less so for other uses. Search traffic is low volume. Transaction traffic is moderate but grows rapidly. Table 7-3 provides details about these workload patterns.

Table 7-3 Customer self-service site workload pattern

Workload pattern	**Customer self-service**
Categories/examples	▶ Home banking ▶ Package tracking ▶ Travel arrangements
Content	▶ Data is in legacy applications ▶ Multiple data sources, requirement for consistency
Security	▶ Privacy ▶ Non-repudiation ▶ Integrity ▶ Authentication ▶ Regulations
Percent secure pages	Medium
Cross-session information	Yes
Searches	▶ Structured by category ▶ Low volume
Unique items	Low
Data volatility	Low
Volume of transactions	Moderate and growing
Legacy integration/complexity	High
Page views	Moderate to low

Online trading (user to business)

Of all site types, online trading sites have the most volatile content, the potential for the highest transaction volumes (with significant swing), the most complex transactions, and the most time sensitive data. Auction sites are characterized by highly dynamic bidding against items with predictable life times.

Trading sites are tightly connected to the legacy systems. Nearly all transactions interact with the back-end servers. Security considerations are high, equivalent to online shopping, with an even larger number of secure pages. Search traffic is low volume. Table 7-4 provides details about these workload patterns.

Table 7-4 Online trading site workload pattern

Workload pattern	**Online trading**
Categories/examples	► Online stock trading ► Auctions
Content	► Extremely time sensitive ► High volatility ► Multiple suppliers, multiple consumers ► Transactions are complex and interact with back end
Security	► Privacy ► Non-repudiation ► Integrity ► Authentication ► Regulations
Percent secure pages	High
Cross-session information	Yes
Searches	► Structured by category ► Low volume
Unique items	Low to medium
Data volatility	High
Volume of transactions	High to very high (very large swings in volume)
Legacy integration/complexity	High
Page views	Moderate to high

Business to business

These sites include dynamic programmatic links between arm's length businesses (where a trading partner agreement might be appropriate). One business is able to discover another business with which it might want to initiate transactions.

In supply chain management, for example, data comes largely from legacy applications and often from multiple sources, thereby exposing data consistency. Security requirements are equivalent to online shopping. Transaction volume is moderate but growing. Transactions are typically complex and connect multiple suppliers and distributors. Table 7-5 provides details about these workload patterns.

Table 7-5 Business-to-business site workload pattern

Workload pattern	**Business to business**
Categories/examples	e-procurement
Content	► Data is in legacy applications ► Multiple data sources, requirement for consistency ► Transactions are complex
Security	► Privacy ► Non-repudiation ► Integrity ► Authentication ► Regulations
Percent secure pages	Medium
Cross-session information	Yes
Searches	► Structured by category ► Low to moderate volume
Unique items	Moderate
Data volatility	Moderate
Volume of transactions	Moderate to low
Legacy integration/complexity	High
Page views	Moderate

Workload characteristics

Your site type will become clear as it is evaluated using Table 7-6 for other characteristics related to transaction complexity, volume swings, data volatility, security, and so on. If the application has further characteristics that could potentially affect its scalability, then add those characteristics to Table 7-6.

Note: All site types are considered to have high volumes of dynamic transactions. This might or might not be a standard consideration when performing sizing and site classification.

Table 7-6 Workload characteristics and Web site classifications

Workload characteristics	Publish/ subscribe	Online shopping	Customer self- service	Online trading	Business to business	Your workload
Volume of user-specific responses	Low	Low	Medium	High	Medium	
Amount of cross-session information	Low	High	High	High	High	
Volume of dynamic searches	Low	High	Low	Low	Medium	
Transaction complexity	Low	Medium	High	High	High	
Transaction volume swing	Low	Medium	Medium	High	High	
Data volatility	Low	Low	Low	High	Medium	
Number of unique items	High	Medium	Low	Medium	Medium	
Number of page views	High	Medium	Low	Medium	Medium	
Percent of secure pages (privacy)	Low	Medium	Medium	High	High	
Use of security (authentication, integrity, non-repudiation)	Low	High	High	High	High	
Other characteristics						High

7.2.4 Determining the most affected components

This step involves mapping the most important site characteristics to each component. Once again, from a scalability viewpoint, the key components of the infrastructure are the load balancers, the application servers, security services, transaction and data servers, and the network. Table 7-7 specifies the significance of each workload characteristic to each component. As the table shows, the effect on each component is different for each workload characteristic.

Table 7-7 Load impacts on components

Workload characteristics	**Web present. server**	**App. server**	**Network**	**Security servers**	**Fire-walls**	**Existing busi-ness servers**	**Data-base server**
High % user-specific responses	Low	High	Low	Low	Low	High	High
High % cross-session information	Med	High	Low	Low	Low	Low	Med
High volume of dynamic searches	High	High	Med	Low	Med	Med	High
High transaction complexity	Low	High	Low	Med	Low	High	High
High transaction volume swing	Med	High	Low	Low	Low	High	High
High data volatility	High	High	Low	Low	Low	Med	High
High # unique items	Low	Med	Low	Low	Low	High	High
High # page views	High	Low	High	Low	High	Low	Low
High % secure pages (privacy)	High	Low	Low	High	Low	Low	Low
High security	High	High	Low	High	Low	High	Low

7.2.5 Selecting the scaling techniques to apply

Best efforts in collecting the information that is needed are worthwhile in order to make the best possible scaling decisions. Only when the information gathering is as complete as it can be, then it is time to consider matching scaling techniques to components. Manageability, security, and availability are critical factors in all design decisions. You should not use techniques that provide scalability but that compromise any of these critical factors.

The scaling techniques are:

- Using a faster machine
- Creating a cluster of machines
- Using appliance servers
- Segmenting the workload
- Using batch requests
- Aggregating user data
- Managing connections
- Using caching techniques

Note: Instead of buying hardware that can handle exponential growth that might not occur, consider these specific approaches:

- For application servers, add more machines. It is appropriate to start with the expectation of more than one application server. Depending on the topology, as discussed in Chapter 9, "Topologies" on page 247, you can assume a load balancer in front of the Web tier or possibly in front of the application server tier. Adding more machines then becomes easier and far less disruptive.
- For data servers, get a server that is initially oversized. Some customers run at just 30% capacity. An oversized server avoids the issue where the whole site can only use one data server. Another scaling option when more capacity is needed is to partition the database into multiple servers or to partition the data into multiple smaller servers, taking advantage of distributed parallel processing techniques.
- Just as many sites separate the application server from the database server, so do they separate the Web server from the application server. The front-end Web serving and commerce application functions are placed on less expensive commodity machines. Because these machines are lower cost, a load balancer is generally deployed with these machines.

The following sections describe these scaling techniques in detail.

Using a faster machine

With a faster machine, the goal is to increase the ability to do more work in a unit of time by processing tasks more rapidly. Upgrading the hardware or software results in a faster machine. However, one of the issues is that software capabilities can limit the hardware exploitation and vice versa. Another issue is that due to hardware or software changes, changes to existing system management policies might be needed.

Creating a cluster of machines

With a cluster of machines, the goal is to service more client requests. Parallelism in machine clusters typically leads to improvements in response time. You can use database partitioning to implement more machines running the database, allowing for parallel processing of large queries. Also, system availability is improved due to failover safety in replicas. The service running in a replica can have associated with it state information that must be preserved across client requests and thus should be shared among machines.

State sharing is probably the most important issue with machine clusters and can complicate the deployment of this technique. WebSphere's workload balancing feature uses an efficient data-sharing technique to support clustering. Issues such as additional system management for hardware and software can also be challenging.

WebSphere Application Server Network Deployment V6 provides the WebSphere Edge Components that you can use for Web server load balanced clusters. The clustering concept is discussed in various chapters throughout this book. Chapter 9, “Topologies” on page 247, covers several configurations that use clusters to improve performance and scalability.

Using appliance servers

With appliance servers, the goal is to improve the efficiency of a specific component by using a special-purpose machine to perform the required action. These machines tend to be dedicated machines that are very fast and optimized for a specific function. Examples are network appliances and routers with cache.

Some issues to consider regarding special machines are the sufficiency and stability of the functions and the potential benefits in relation to the added complexity and manageability challenges. Make sure to apply the same resiliency techniques to these devices. Do not introduce an appliance that creates a new single point of failure in the environment.

Segmenting the workload

With segmenting the workload, the goal is to split the workload into manageable chunks and thereby obtaining a more consistent and predictable response time. The technique also makes it easier to manage on which servers the workload is placed. Combining segmentation with replication often offers the added benefit of an easy mechanism to redistribute work and scale selectively as business needs dictate.

An issue with this technique is that to implement the segmentation, the different workloads serviced by the component need to be characterized. You should take care during implementation to ease the characterization of the workload. After segmenting the workload, additional infrastructure might be required to balance physical workload among the segments. Refer to Chapter 9, “Topologies” on page 247 for further details about workload segmentation and balancing configurations.

Using batch requests

With batch requests, the goal is to reduce the number of requests that are sent between requesters and responders (such as between tiers or processes) by allowing the requester to define new requests that combine multiple requests. The benefits of this technique arise from the reduced load on the responders by eliminating overhead associated with multiple requests. It also reduces the latency experienced by the requester due to the elimination of the overhead costs with multiple requests. These requests can be processed offline or at night time to reduce the load during peak hours.

One of the issues with this technique is that there can be limits in achieving reuse of requests due to inherent differences in various requests types (for example, Web front end differs from voice response front end). This can lead to increased costs of supporting different request types. Another issue is the latency associated with batch applications. There can also be regulations against delay of delivery of that information that restrict the ability to batch requests.

Aggregating user data

With aggregation of user data, the goal allows rapid access to large customer data that is controlled by existing system applications and support personalization that is based on customer specific data. See 2.6.1, “Data consolidation” on page 43 for information about the nature of data aggregation.

Accessing existing customer data that is spread across multiple legacy system applications can cause these applications to become overloaded, especially when the access is frequent. This issue can degrade response time. To alleviate this issue, the technique calls for aggregating customer data into a customer information service (CIS). A CIS that is kept current can provide rapid access to the customer data for a very large number of customers. Thus, it can provide the required scalability. Another possible tool is the use of a front-end software that maps all the back end data as a single store, though in native format. A product like DB2 Information Integrator provides such a facility.

An issue with a CIS is that it needs to scale very well to support large data as well as to field requests from a large number of application servers (requesters). Data currency issues also can result from creation of a CIS. An issue associated with DB2 Information Integrator type tools is that not all proprietary formats for data are supported. These proprietary formats might be necessary to develop your own maps, depending on the nature of the legacy application.

Managing connections

With connection management, the goal is to minimize the number of connections that are needed for an end-to-end system, as well as to eliminate the overhead of setting up connections during normal operations. To reduce the overhead associated with establishing connections between each layer, a pool of pre-established connections is maintained and shared among multiple requests flowing between the layers.

For instance, WebSphere Application Server provides database connection managers to allow connection reuse. It is important to note that a session might use multiple connections to accomplish its tasks, or many sessions can share the same connection. This is called connection pooling in the WebSphere connection manager.

Another pool of connections is at the Web container level. The application server environment pre-allocates this pool to allow for quick access to requests. After a request has been satisfied, the container releases the connection back to the pool. There are configuration options for allowing this pool to grow beyond maximum and to limit how far beyond. These options, however, can create additional memory requirements. You should take care when adjusting the parameters that are related to the Web container thread pool.

The key issue with this technique is with maintaining a session's identity when several sessions share a connection. Reusing existing database connections conserves resources and reduces latency for application requests, thereby helping to increase the number of concurrent requests that can be processed. Managing connections properly can improve scalability and response time. Administrators must monitor and manage resources proactively to optimize component allocation and use.

Refer to Chapter 14, “Planning for performance, scalability, and high availability” on page 405 for more information.

Using caching techniques

With caching techniques, the goal is to improve performance and scalability by reducing the length of the path traversed by a request and the resulting response and by reducing the consumption of resources by components.

Caching techniques can be applied to both static and dynamic Web pages. A powerful technique to improve performance of dynamic content is to identify and generate asynchronously Web pages that are affected by changes to the underlying data. After these changed pages are generated, they must be effectively cached for subsequent data requests.

There are several examples of intelligent caching technologies that can significantly enhance the scalability of e-business systems. For static Web pages, the key is to determine the appropriate content expiration, so that if a site changes, the caching component knows to refresh the cached copy.

The key issue with caching dynamic Web pages is determining what pages should be cached and when a cached page has become obsolete. For information regarding caching techniques, see Chapter 14, “Planning for performance, scalability, and high availability” on page 405.

At the database layer, caching should be used based on what facilities are provided by the Database Management System. Using data pre-fetching, you can have the database read additional data with the expectation that this data will also be needed very soon. This can increase performance by minimizing the number of disk reads that are required. Also, memory buffers can be used to house data pages when read into memory, reducing disk access. The key is to make sure the system has adequate main physical memory to provide to the database.

7.2.6 Applying the techniques

When performing any scaling, tuning, or adjustments to the application environment, you should first try to apply the selected technique or techniques to a test environment to avoid direct impact on the production environment. The idea is to evaluate not only the performance and scalability impact to a component but also to determine how each change affects the surrounding components and the end-to-end infrastructure. A situation where improvements on one component result in an increased (and unnoticed) load on another component is undesirable.

Figure 7-2 on page 207 illustrates the typical relationship between the techniques and the key infrastructure components. Using this figure, you can identify the key techniques for each component.

In many cases, all techniques cannot be applied because one or more of the following statements are true:

- Investing in the techniques, even if it would prove helpful, is not affordable.
- There is no perceived need to scale as much as the techniques allow.
- A cost/benefit analysis shows that the technique will not result in a reasonable payback.

Therefore, there must be a process for applying these techniques in different situations so that the best return is achieved. This mapping is a starting point and shows the components to focus on first, based on application workload.

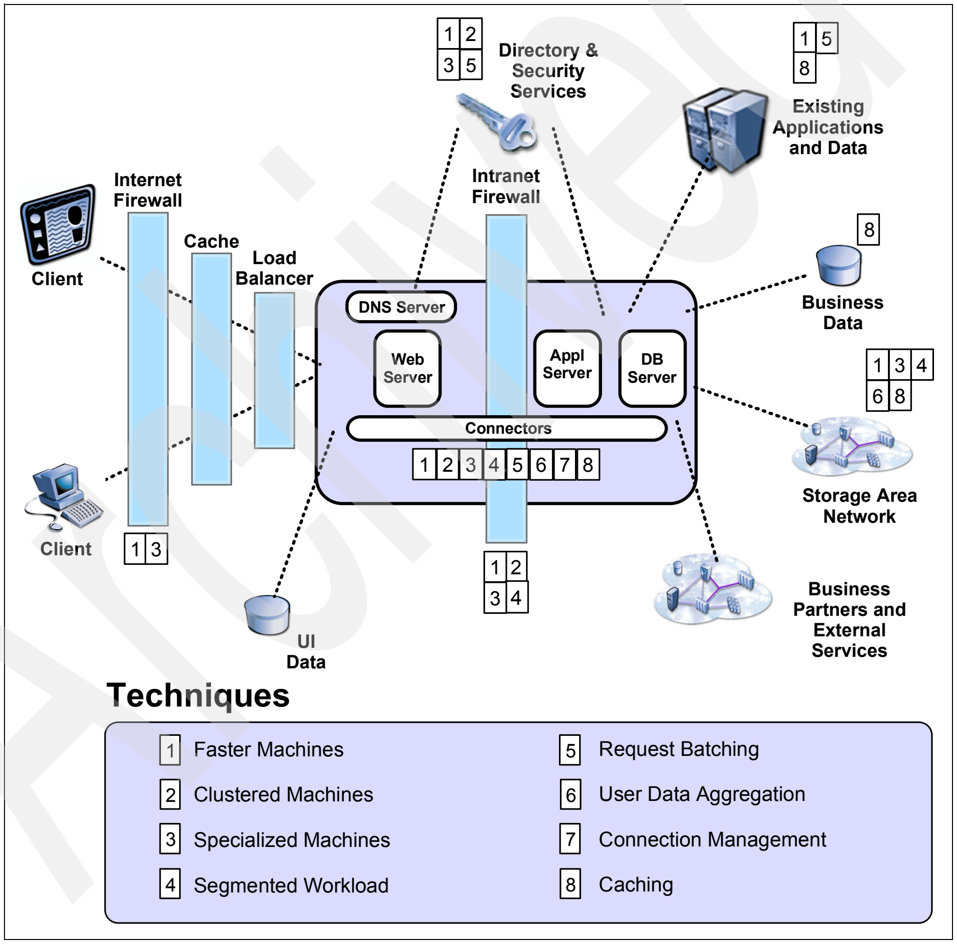


Figure 7-2 Scaling techniques applied to components

7.2.7 Re-evaluating

As with all performance-related work, tuning is required. The goals are to eliminate bottlenecks, scale to a manageable status for those that cannot be eliminated, and work around those that cannot be scaled. Some examples of tuning actions and the benefits that are realized are:

- Increasing Web server threads raises the number of requests that could be processed in parallel.
- Adding indexes to the data server reduces I/O bottlenecks.
- Changing the defaults for several operating system variables allows threaded applications to use more heap space.
- Caching significantly reduces the number of requests to the data server.
- Increasing the number of edge/appliance servers improves load balancing.
- Upgrading the data server increases throughput.

Such results demonstrate the potential benefits of systematically applying scaling techniques and continuing to tune. For an introduction to WebSphere tuning, see Chapter 14, "Planning for performance, scalability, and high availability" on page 405.

Recognize that any system is dynamic. The initial infrastructure will at some point need to be reviewed and possibly grown. Changes in the nature of the workload can create a need to re-evaluate the current environment. Large increases in traffic will require examination of the machine configurations. As long as you understand that scalability is not a one time design consideration, that instead it is part of the growth of the environment, you will be able to keep a system resilient to changes and avoid possibly negative experiences due to poorly planned infrastructure.

For more details about the approach used in this section, refer to the Design for Scalability article available at:

http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/scalability.html

7.3 Sizing

After creating a draft of your initial design, you should have built some scalability into the architecture. However, you still need to know the number of machines that you will have to work with for the project.

You might not have your application finished, which is the case for most projects. For example, the overall project budget has to be approved before the development project is started, or the developers do not anticipate having the application ready for some time.

Thus, it is imperative that you have as static a version of your application design at this point, if possible. You should work closely with your development team, if at all possible. The better you understand the application, the better your sizing estimation is going to be.

At this point, you should also consider which hardware platform you want to deploy and if you prefer either scaling-up or scaling-out. The hardware platform decision is primarily dependent on your platform preference, which platforms have sizing information available and which platforms WebSphere Application Server supports. Hardware decisions might also be driven by the availability of hardware that forces a limitation in the operating systems that can be deployed.

The decision between scaling-up or scaling-out is usually a decision of preference and cost for your environment. However, the reality is that application resiliency issues can change your preferences. Scaling-up means to do vertical scaling on a small number of machines with many processors. This can present fairly significant single points of failure. Be aware that the potential fails in an environment that is composed of fewer large machines. Scaling-out, on the other hand, means using a larger number of smaller machines. This can generally be thought of as significantly more resilient, because it is unlikely that the failure of one small server is adequate to create a complete application outage.

What you need to understand, though, is that the sizing estimations are solely based on your input, which means the better the input, the better the results. The sizing work assumes an average application performance behavior. There are no tests being run, but an average response time is assumed for each transaction and calculations are performed to determine the estimated number of machines and processors your application will require. If your enterprise has a user experience team, they might have documented standards for a typical response time that your new project will be required to meet.

If you need a more accurate estimation of your hardware requirements and you already have your application, you might want to consider one of the benchmarking services discussed in 7.4, “Benchmarking” on page 210.

Notice that this sizing is for your production environment only. Based on this estimation, you do not only have to update your production implementation design but you also have to update the designs for the integration and development environments accordingly. The key is to remember that changes to the production environment should be incorporated into the development and testing environments as well, if cost considerations do not merit otherwise.

7.4 Benchmarking

Benchmarking is the process used to take an application environment and determine capacity of that environment through load testing. This determination allows you to make reasonable judgements as your environment begins to change. Using benchmarks, you are able to determine the current work environment capacity and set expectations as new applications and components are introduced.

Benchmarking is primarily interesting to two kinds of clients:

- Clients who already have an application and want to migrate to a new version of WebSphere or who want to evaluate the exact number of machines for their target deployment platform.
- Clients who sell products that are based on WebSphere and who want to provide sizing estimations for their products.

Many sophisticated enterprises maintain a benchmark of their application stack and change it after each launch or upgrade of a component. These customers usually have well-developed application testing environments and teams dedicated to the cause. For those that do not, alternatives are available such as the IBM Test Center. There are also third-party benchmark companies that provide this service. When choosing, make sure that the team that performs the benchmark tests has adequate knowledge of the environment and a clear set of goals. This helps to reduce the costs of the benchmark tests and creates results that are much easier to quantify.

7.4.1 IBM Test Center

IBM Global Services offers you the ability to retain IBM for Performance Management, Testing, and Scalability services. This team will come to a customer site and assess the overall site performance. This investigation is platform neutral, with no sales team poised to sell additional hardware as a result of the analysis. Offerings include, but are not limited to:

- Testing and Scalability Services for TCP/IP networks
- Testing and Scalability Services for Web site stress analysis
- Performance Engineering and Test Process Consulting
- Performance Testing and Validation
- Performance Tuning and Capacity Optimization

When using a service like those provided by the IBM Test Center, you are presented with large amounts of supporting documentation and evidence to support the results of the tests. You can then use this data to revise your current architecture or possibly just change the overall infrastructure footprint to add additional machines or to correct single points of failure.

These offerings can be purchased through your local IBM account representative.

7.5 Performance tuning

Performance is one of the most important non-functional requirements for any WebSphere environment. Application performance should be tracked continuously during your project.

Imagine your project is finished, you switch your environment to production, and your environment is unable to handle the user load. This is by far the most user-visible problem that you could have. Most users are willing to accept small functional problems when a system is rolled out, but performance problems are unacceptable to most users and affect everyone working on the system.

7.5.1 Application design problems

Many performance problems cannot be fixed by using more hardware or changing WebSphere parameters. Thus, you really want to make performance testing (and tuning) part of your development and release cycles. Otherwise, it takes much more effort and money to correct issues after they occurred in production than to fix them up front. If performance testing is part of your development cycle, you are able to correct issues with your application design much earlier, resulting in fewer issues when using your application on the production environment.

7.5.2 Understand your requirements

Without a clear understanding of your requirements, you have no target that you can tune against. It is important when doing performance tuning to know your objectives. Do not waste time trying to do performance tuning on a system that was improperly sized and cannot withstand the load, no matter how long you tune it. Also, do not continue tuning your system when you are already beyond your performance targets.

Understanding your requirements demands knowledge of two things:

- Non-functional requirements
- Sizing

If you do not have this information and you are asked to tune a system, you will either fail or will not know when to stop tuning.

7.5.3 Test environment setup

When performing any performance tests, follow these general tips throughout all your tests:

- Perform your tests on machines that mirror the production server state. Be as much realistic as possible with the test environment. Try to get the closest hardware configuration possible to the production environment. Make sure you are able to extrapolate the test results to the production environment.
- Make sure that nobody is using the test machines and that no background processes are running that consume more resources than what you find in production. As an example, if the intent is to test performance during the database backup, then make sure the backup is running.

  For example, it is OK to run monitoring software in the background, such as Tivoli or BMC, which will run in production. However, having a database backup running might not be valid for your desired test.
- Check for processor, memory, and disk utilization before and after each test run to see if there are any unusual patterns. If the target environment will be using shared infrastructure (messaging servers or authentication providers, for example) try to make sure the shared component is performing under projected shared load.
- Isolate network traffic as much as possible. Using switches, there is rarely a circumstance where traffic from one server overruns the port of another. It is possible, however, to flood ports used for routing off the network to other networks or even the switch backbone for very heavy traffic. Make sure that your network is designed in a manner that isolates the testing environment as much as possible prior to starting, as performance degradation of the network can create unexpected results.

7.5.4 Integration testing

Before putting a new release into production, you want to know if the new version will really work as it is supposed to. To do this, you need an integration system.

An integration system should be a very close approximation to the logical layout of the production environment. This could mean multiple partitioned machines to simulate the numbers of production machines, or it could mean smaller machines in terms of processing but equal in number. The following are general recommendations that you might want to follow:

- For a production system with up to two machines in a single layer, you should either get the same machine for integration testing or half the processors in each machine.

- If you have more than two clustered machines in a single layer, you might want to get half the number of machines than you have in production.

It is important to keep in mind that the integration system should be at least half as large as your production environment to get meaningful results and you want to have the same workload manager characteristic that you have in production. This is necessary because WebSphere Application Server behaves differently in a cluster than with a single application server. If using persistent session state, this is an essential item for testing. Again, try to be as realistic as possible ensuring that you can extrapolate the test results to the production environment.

During the production phase of your application, you use this environment to perform load tests and compare them to baseline results that you captured with your earlier software builds. Based on this, because your production environment is some multiple of the size of your integration environment, you will be able to tell how the new release will impact your production environment and whether this release is already fit for being deployed there.

7.5.5 Development testing

During the application development phase, the development team should publish regular code base versions and builds to a test server. This activity supports incremental code development, integration, and testing. The development team should also create a Build Verification Test process, one where each new build is executed before making this build available to the team. A Build Verification Test covers one or more test cases or scenarios, and it should test the critical paths through the code that are operational and require validation. Executing the Build Verification Test ensures that the new build has no obvious errors.

The Build Verification Test process should start as soon as one or more use cases are ready and has been tested on the test server.

The back-end test system where the Build Verification Test is carried out should reasonably mimic the interfaces and communication systems of the production systems to ensure the code can be tested using end-to-end scenarios. The better able the development team is to create an accurate test environment, the more effective and thorough its tests will be.

Following the application development phase is the application test phase. This phase is entirely dedicated to testing and adds value by ensuring that all code is tested at the site level before the site itself becomes fully operational. Unless otherwise indicated, all stress and performance tests should be performed on the actual site code.

The main activities during the application test phase are:

- Functional Verification Test (FVT).

  The main objective of this activity is to test the functionality of the code from end-to-end, using a suite of test cases.
- System Integration Test (SIT).

  This activity is a preparatory phase that determines if the customer test environment is ready to support system tests and User Acceptance Test.
- System Verification Test (SVT).

  The purpose of this activity is to validate the functionality of site code received from the development team. All system tests should be performed in the simulated environment.
- User Acceptance Test (UAT).

  This activity focuses on the final site look-and-feel, how the site flows and behaviors, and overall business process validation. Normally, the user is the final consumer — the customer — the one who carries out this activity.

Each activity focuses on a different aspect of the application testing phase, performance should be monitored all along. The details of these activities are beyond the scope of this book.

7.5.6 Load factors

The most important factors that determine how you conduct your load tests are:

- Request rate
- Concurrent users
- Usage patterns

This is not a complete list and other factors can become more important depending on the kind of site that is being developed. These factors are explained in greater detail in Chapter 14, “Planning for performance, scalability, and high availability” on page 405.

Usage patterns

At this point in the project, it is very important that you think about how your users will use the site. You might want to use the use cases that your developers defined for their application design as an input to build your usage patterns. This makes it easier to build the scenarios later that the load test would use.

Usage patterns consist of:

- Use cases modeled as click streams through your pages.
- Weights applied to your use cases.

Combining weights with click streams is very important because it shows you how many users you expect in which of your application components and where they generate load. After all, it is a different kind of load if you expect 70% of your users to search your site instead of browsing through the catalog than the other way around. These assumptions also have an impact on your caching strategy.

Make sure that you notify your developers of your findings so that they can apply them to their development effort. Make sure that the most common use cases are the ones where most of the performance optimization work is performed.

To use this information later when recording your load test scenarios, we recommend that you write a report with screen shots or URL paths for the click streams (user behavior). Include the weights for your use cases to show the reviewers how the load was distributed.

7.5.7 Production system tuning

This is the only environment that impacts the user experience for your customers. This is where you apply all the performance, scalability, and high availability considerations that are described in Chapter 14, “Planning for performance, scalability, and high availability” on page 405.

Tuning this system is an iterative process and, instead of testing multiple versions here and comparing them, you are changing the WebSphere parameters themselves and optimizing them to suit your runtime environment.

Important: To perform tuning in the production environment, you should already have the final version of code that runs there. This version should have passed performance tests on the integration environment prior to changing any WebSphere parameters on the production system.

When changing a production environment, you should use regular system administration practices, such as those outlined in Chapter 13, “Planning for system management” on page 371. This implies some standard practices:

- Change only one parameter at a time.
- Document all your changes.
- Compare several test runs to the baseline.

These test runs should not differ by more than a small percent or you have some other problem with your environment that you need to sort out before you continue tuning.

As soon as you finished tuning your production systems you would apply the settings, where it makes sense, to your other environments to make sure they are similar to production. You should plan to re-run your tests there to establish new baselines on these systems and to see how these changes affect the performance.

Keep in mind that usually you only have one chance on getting this right. Normally, as soon as you are in production with your system, you cannot run performance tests on this environment any more - simply because you cannot take the production system offline to run more performance tests. If a production system is being tested, it is likely that the system is running in a severely degraded position, and you have already lost half the battle.

Note: Because it is rare to use a production system for load tests, it is usually a bad idea to migrate these environments to new WebSphere versions without doing a proper test on an equivalent test system or new hardware.

After completing your first performance tests on your production systems and tuning the WebSphere parameters, you should evaluate your results and compare them to your objectives to see how all of this worked out for you.

7.5.8 Conclusions

There are various possible outcomes from your performance tests that you should clearly understand and act upon:

- Performance meets your objectives.

  Congratulations! However, do not stop here. Make sure that you have planned for future growth and that you are meeting all your performance goals. After that, we recommend documenting your findings in a performance tuning report and archiving it. Include all the settings that you changed to reach your objectives.

  This report comes in handy when you set up a new environment or when you have to duplicate your results somewhere else on a similar environment with the same application. This data also is essential when adding additional replicas of some component in the system, as they need to be tuned to the same settings the current resources use.

- Performance is slower than required.

  Your application performance is somewhat slower than expected, and you have already done all possible application and WebSphere parameter tuning. You might need to add more hardware (for example, increase memory, upgrade processors, and so forth) to those components in your environment that showed to be bottlenecks during your performance tests. Then, run the tests again. Verify with the appropriate teams that there were no missed bottlenecks in the overall system flow.

- Performance is significantly slower than required.

  In this case, you should start over with your sizing and ask the following questions:

  – Did you underestimate any of the application characteristics during your initial sizing? If so, why?

  – Did you underestimate the traffic and number of users/hits on the site?

  – Is it still possible to change parts of the application to improve performance?

  – Is it possible to obtain additional resources?

  After answering these questions, you should have a better understanding about the problem that you face right now. Your best bet is to analyze your application and try to find the bottlenecks that cause your performance problems. Tools like the Profiler that is part of Rational Application Developer can help you with this. For further information about Profiling tools refer to *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392.

8

Topology selection criteria

Topology refers to what devices and computers are going to be used to set up a Web application — the physical layout of each one and the relationship between them. When selecting a topology, there are a number of considerations that impact the decision of which one to use. This chapter covers the key concepts that affect the topology selection as well as the terminology used across topologies. This chapter contains the following sections:

- Common terminology
- WebSphere Application Server terminology
- Security
- Performance
- Throughput
- Availability
- Maintainability
- Session management
- Topology selection summary

The terms and concepts that are defined in this chapter are complemented when reviewing the different topologies in Chapter 9, “Topologies” on page 247.

8.1 Common terminology

Before explaining each topology, it is key to have a full understanding of the common terminology for topologies in general, Web application environments, and the specific terminology for WebSphere Application Server V6.

A common terminology refers to a terminology that is common to all the topologies covered in Chapter 9, “Topologies” on page 247. There are many other terms that are incorporated in the discussion in later chapters in this book.

8.1.1 Web application server node

A Web application server node is an application server that includes a Web server and that is typically designed to be accessed by Web clients and to host both presentation and business logic.

The Web application server node is a functional extension of the informational (publishing-based) Web server. It provides the technology platform and contains the components to support access to both public and user-specific information by users employing Web browser technology. For the latter, the node provides robust services to allow users to communicate with shared applications and databases. In this way, it acts as an interface to business functions, such as banking, lending, and human resources systems.

The node can contain these data types:

- HTML text pages, images, multimedia content to be downloaded to the HTTP client (Web browser), and XML files
- Servlets, JavaServer Pages (JSPs), and JavaServer Faces (JSFs)
- Enterprise beans
- Application program libraries, such as Java applets for dynamic download to client workstations

In these topologies the Web application server node is implemented using one (or both) of the following:

- WebSphere Application Server and the embedded HTTP transport
- WebSphere Application Server, a supported Web server (for example, IBM HTTP Server) and the Web server plug-in

8.1.2 Web server redirector node and application server node

Most topologies feature a stand-alone Web server that resides in a DMZ between two firewalls. To separate the Web server function from the Web application server function, we effectively split the Web application server into two new nodes: a Web server redirector node and an application server node.

Web clients send requests to the Web server redirector node, which serves static content such as HTTP pages. Requests for dynamic content, that requires processing by servlets, JSPs, enterprise beans, and back-end applications are forwarded to the application server.

In the topologies that this section discusses, the Web server redirector is implemented using a supported Web server (IBM HTTP Server) and the appropriate Web server plug-in. The application server node is implemented using WebSphere Application Server.

8.1.3 Domain and protocol firewalls

A firewall is a hardware and software system that manages the flow of information between the Internet and an organization's private network. Firewalls can prevent unauthorized Internet users from accessing private networks that are connected to the Internet, especially intranets, and can block some virus attacks, as long as those viruses are coming from the Internet. Also can prevent denial of service attacks.

A firewall can separate two or more parts of a local network to control data exchange between departments. Components of firewalls include filters or screens, each of which controls transmission of certain classes of traffic. Firewalls provide the first line of defense for protecting private information, but comprehensive security systems combine firewalls with encryption and other complementary services, such as content filtering and intrusion detection.

Firewalls control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- Screening routers (the protocol firewall)

  Prevents unauthorized access from the Internet to the DMZ. The role of this node is to provide the Internet traffic access only on certain ports and to block other IP ports.

- Application gateways (the domain firewall)

  Prevents unauthorized access from the DMZ to an internal network. The role of firewall allows the network traffic originating from the DMZ and not from the Internet. It also provides some filtering from the intranet to the DMZ. A pair of firewall nodes provides increasing levels of protection at the expense of increasing computing resource requirements. The protocol firewall is typically implemented as an IP router.

8.1.4 Directory and security services node

The directory and security services node supplies information about the location, capabilities, and attributes (including user ID and password pairs and certificates) of resources and users known to this Web application system. This node can supply information for various security services (authentication and authorization) and can also perform the actual security processing, for example, to verify certificates. The authentication in most current designs validates the access to the Web application server part of the Web server, but this node also authenticates for access to the database server.

An example of a product that provides directory services is IBM Tivoli Directory Server included in WebSphere Application Server Network Deployment V6 package.

8.1.5 Web presentation server node

The Web presentation server node provides services to enable a unified user interface. It is responsible for all presentation related activity. In its simplest form, Web presentation server node serves HTML pages and runs servlets and JSPs. For more advanced patterns, it acts as a portal and provides the access integration services (single signon, for example). It interacts with the personalization server node to customize the presentation based on the individual user preferences or on the user role. The Web presentation server allows organizations and their users to standardize and configure the presentation of applications and data in the most efficient way, while enabling fine-grained access control.

8.1.6 Database server node

The function of the database server node is to provide a persistent and reliable data storage and retrieval in support of the user-to-business transactional interaction. The data that is stored is relevant to the specific business interaction, for example bank balance, insurance information, and current purchase by the user.

It is important to note that the mode of database access is perhaps the most important factor determining the performance of this Web application, in all but the simplest cases. The recommended approach is to collapse the database accesses into single or very few calls. One approach for achieving this is by coding and invoking stored procedure calls on the database.

8.1.7 Load balancer node

The load balancer node, also called IP sprayer, provides horizontal scalability by dispatching HTTP requests among several identically configured Web server or Web server redirector nodes. In these topologies, the load balancer node is implemented using the Edge Components provided with the WebSphere Application Server Network Deployment V6 installation package.

For information about the terms mentioned in this section and new terms that will be addressed next, see 9.10, “Horizontal scaling with IP sprayer topology” on page 278.

8.2 WebSphere Application Server terminology

The terminology and definitions referenced in this section are specific to WebSphere Application Server environments and are used throughout this book.

8.2.1 Plug-in module

The WebSphere plug-in is a component of WebSphere Application Server that is installed on the Web server, which routes requests from the Web server to the application server. It knows if a request can be served by the Web server or if it has to be routed to the application server. It also provides load balancing and failover support when configured to route requests from a Web server to more than one application server.

8.2.2 Plug-in configuration file

The plug-in configuration file is an XML file that has the information the plug-in module needs to properly route requests from the Web server to the application server (plugin-cfg.xml).

8.2.3 Application Server Node

An Application Server Node (or Node) is a grouping of application servers for configuration and operational management on one machine. You can have multiple Application Server Nodes on a single machine depending on the configuration.

8.2.4 Node Agent

The Node Agent identifies the application servers within the Application Server Node in order to be managed by the Deployment Manager. It is purely an administrative agent and is not involved in application serving functions. Depending on the configuration, you can have multiple Node Agents on one single physical machine but only one per Application Server Node.

Node Agent is created under the covers when you add (federate) a stand-alone Node to a Cell.

8.2.5 Deployment Manager

The Deployment Manager manages one or more Application Server Nodes in a distributed topology (Cell). It communicates (for configuration and administration) with the Application Server Nodes through the Node Agents.

8.2.6 Cell

A Cell is a network of multiple Application Server Nodes. It provides a single logical administration domain and contains only one Deployment Manager. Deployment Managers and Nodes can be on the same or different machines and logical partition (LPAR).

Note: Another way to describe a Cell is by identifying all the Node Agents managed by a single Deployment Manager.

8.2.7 Cluster

A Cluster is a grouping of application servers called Cluster Members. These Cluster Members share the same set of J2EE applications. A cluster provides scalability and failover support. A Cell can have zero or more Clusters. It can span machine or LPAR boundaries (vertical scaling and horizontal scaling) and can span to different operating systems (but it cannot span from distributed to z/OS).

8.2.8 Cluster Member

A Cluster Member is an application server that runs the same set of J2EE applications inside a Cluster. In other words, those Cluster Members within the same Cluster will be running the same application in parallel. The Cluster Members are coordinated by the Deployment Manager through the Node Agent.

8.2.9 Web server Managed Node

A Web server Managed Node is a Web server that is being managed by the Deployment Manager. It provides the ability to start and stop the Web server and to push the plug-in configuration file to the Web server automatically. The Web server Managed Node requires an Application Server Node to be created on the Web server machine. It is commonly used when Web servers are installed behind a firewall where an Application Server Node can be installed.

8.2.10 Web server Unmanaged Node

A Web server Unmanaged Node is a Web server that configuration is not being managed by any Deployment Manager. It is commonly used when Web servers are installed outside of a firewall where no Application Server Node can be installed. All Web server implementations prior to V6 use Unmanaged Nodes.

8.2.11 WebSphere Profile

The concept of WebSphere Profile was introduced in V5 and was called WebSphere Instance in that release. Each WebSphere Profile has its own user data that includes WebSphere configuration. You can create WebSphere Application Server profiles (from an installation of WebSphere Application Server) or WebSphere Application Server Deployment Manager profiles (from an installation of WebSphere Application Server Network Deployment V6).

Each profile uses a separate configuration file but shares the binaries from which it was created. Each profile is distinguished by its base path, its own directory structure, and its own setupCmdLine script to configure its command-line environment.

8.2.12 Managed Node

A Managed Node is a Node that has an application server and a Node Agent that belongs to a Cell.

8.2.13 Node Group

A Node Group is a collection of Managed Nodes. A Node Group defines a boundary for server Cluster formation. Nodes that you organize into a Node Group should be similar in terms of installed software, available resources, and configuration to enable servers on those Nodes to host the same applications as part of a server Cluster. The Deployment Manager does not validate the Nodes in a Node Group.

Node Groups are optional and are established at the discretion of the WebSphere Application Server administrator. However, a Node must be a member of a Node Group. Initially, all application server Nodes are members of the default Node Group named DefaultNodeGroup. In addition, a Node can be a member of more than one Node Group. However, a Node on a distributed platform and a Node on a z/OS platform cannot be members of the same Node Group.

On the z/OS platform, an Application Server Node must be a member of a sysplex Node group. Nodes in the same sysplex must be in the same sysplex Node group. A Node can be in one sysplex node group only.

To delete a Node Group, the Node Group must be empty. The default Node Group cannot be deleted.

8.2.14 High availability manager

WebSphere Application Server uses a high availability manager to eliminate single points of failure. A high availability manager is responsible for running key services on available application servers rather than on a dedicated one (such as the Deployment Manager). It takes advantage of fault tolerant storage technologies such as network attached storage (NAS), which significantly lowers the cost and complexity of high availability configurations. The high availability manager also provides peer-to-peer failover for critical services by always maintaining a backup for these services.

A high availability manager continually monitors the application server environment. If an application server component fails, the high availability manager takes over the in-flight and in-doubt work for the failed server. This action significantly improves application server availability.

In a highly available environment, all single points of failure are eliminated. Because the high availability manager function is dynamic, any configuration changes that you make and save while an application server is running are eventually picked up and used. You do not have to restart an application server to enable a change. For example, if you change a policy for a messaging engine high availability group while the messaging engine is running, the new policy is dynamically loaded and applied, and the behavior of the messaging engine reflects this change.

A high availability manager focuses on recovery support and scalability in the following areas:

- Messaging
- Transaction managers
- Workload management controllers
- Application servers
- WebSphere partitioning facility instances

To provide this focused failover service, the high availability manager supervises the Java Virtual Machines (JVMs) of the application servers that are core group members. The high availability manager uses one of the following methods to detect failures:

- An application server is marked as failed if the socket fails. This method uses the KEEP_ALIVE function of TCP/IP and is very tolerant of extreme application server loading, which might occur if the application server is swapping or thrashing heavily. This method is recommended for determining a JVM failure if you are using multicast emulation and are running enough JVMs on a single application server to push the application server into extreme processor starvation or memory starvation.
- A JVM is marked as failed if it stops sending heartbeats for a specified time interval. This method is referred to as active failure detection. When it is used, a JVM sends out one heartbeat, or pulse, every second. If the JVM is unresponsive for more than 20 seconds, it is considered down. You can use this method with multicast emulation. However, this method must be used for true multicast addressing.

In either case, if a JVM fails, the application server on which it is running is separated from the core group, and any services running on that application server are failed over to the surviving core group members.

A JVM can be a Node Agent, an application server or a Deployment Manager. If a JVM fails, any singletons running in that JVM are restarted on a peer JVM after the failure is detected. This peer JVM is already running and eliminates the normal startup time, which potentially can be minutes.

All of the application servers in a cell are defined as members of a core group. Each core group has only one logical high availability manager that services all of the members of that core group. The high availability manager is responsible for making the services within a core group highly available and scalable. It continually polls all of the core group members to verify that they are active and healthy.

A policy matching program is used to localize certain policy-driven components and to place these components into high availability groups. When a core group member fails, the high availability manger assigns the failing member's work to the same type of component from the same high availability group. Using NAS devices in the position of common logging facilities helps to recover in-doubt and in-flight work if a component fails.

WebSphere Application Server provides a default core group that is created during installation. New server instances are added to the default core group as they are created. The WebSphere Application Server environment can support multiple core groups, but one core group is usually sufficient for most environments.

8.2.15 Core group

A Core Group is a set of application servers that can be divided up into various high availability groups. It is a statically defined component of the WebSphere Application Server high availability manager function that monitors the application server environment and provides peer-to-peer failover of application server components.

A Core Group can contain one or more high availability groups. However, a high availability group must be contained totally within a single core group. Any component, such as the service integration bus component of IBM service integration technologies or the WebSphere Application Server transaction manager, can create a high availability group for that component's use. For example, the service integration bus might need a high availability group to support its messaging engines, and the transaction manager component might need a high availability group to support its transaction logs.

A Cell must have at least one Core Group. The WebSphere Application Server creates a default Core Group, called DefaultCoreGroup, for each Cell. All the server processes and JVMs, including Node Agents on all Managed Nodes, the Deployment Manager, and application servers residing within a Cell are initially members of this default Core Group.

When properly configured, the default Core Group is sufficient for establishing a high availability environment. However, certain topologies require the use of multiple core groups. For example, if a firewall is used to separate the proxy environment from the server environment, an additional core group is required in order to provide proper failover support. For this particular type of environment, application servers outside of the firewall are members of a separate Core Group from the application servers that are inside of the firewall.

The Core Group contains a bridge service, which supports cluster services that span multiple Core Groups. Core Groups are connected by access point groups. A Core Group access point defines a set of bridge interfaces that resolve to IP addresses and ports. It is through this set of bridge interfaces that the Core Group bridge provides access to a Core Group.

If you create additional core groups when you move core group members to the new core groups, remember that:

- Each server process and JVM, including Node Agents on all Managed Nodes, the Deployment Manager, and application servers within a Cell can only be a member of one Core Group. Core Groups cannot overlap each other.
- If a Cluster is defined for the Cell, all of the Cluster Members must belong to the same Core Group.

Network communication between all the members of a Core Group is essential. The network environment must consist of a fast local area network (LAN) with full Internet Protocol (IP) visibility and bidirectional communication between all Core Group members. IP visibility means that each member is entirely receptive to the communications of any other Core Group member.

8.2.16 High availability groups

High availability groups are dynamically created components of a core group. They cannot be configured directly but are directly affected by static data, such as policy configurations, which are specified at the core group level.

A high availability group cannot extend beyond the boundaries of a core group. However, members of a high availability group can also be members of other high availability groups, as long as all of these high availability groups are defined within the same core group.

Any WebSphere Application Server component, such as the transaction manager, can create a high availability group for that component to use. The component code must specify the attributes that are used to create the name of the high availability group for that component. For example, to establish a high availability group for the transaction manager:

- The code included in the transaction manager component code specifies the attribute type=WAS_TRANSACTIONS as part of the name of the high availability group that is associated with this component.
- The high availability manager function includes the default policy Clustered TM Policy that includes type=WAS_TRANSACTIONS as part of its match criteria.

Whenever transaction manager code joins a high availability group, the high availability manager matches the match criteria of the Clustered TM Policy to the high availability group member name. In this example, the string type=WAS_TRANSACTIONS included in the high availability group name is matched to the same string in the policy match criteria for the Clustered TM Policy. This match associates the Clustered TM Policy with the high availability group that was created by the transaction manager component.

After a policy is established for a high availability group, you can change some of the policy attributes, such as quorum, fail back, and preferred servers. However, you cannot change the policy type. If you need to change the policy type, you must create a new policy and then use the match criteria to associate it with the appropriate group.

If you want to use the same match criteria, you must delete the old policy before defining the new policy. You cannot use the same match criteria for two different policies.

8.2.17 Data replication service

Replication is a service that transfers data, objects, or events among application servers. Data replication service (DRS) is the internal WebSphere Application Server component that replicates data. Session manager uses the data replication service when configured to do memory-to-memory replication. When memory-to-memory replication is configured, session manager maintains data about sessions across multiple application servers, preventing the loss of session data if a single application server fails.

Dynamic cache uses the data replication service to further improve performance by copying cache information across application servers in the cluster, preventing the need to repeatedly perform the same tasks and queries in different application servers.

Stateful session beans use the replication service so that applications using stateful session beans are not limited by unexpected server failures.

You can define the number of replicas that DRS creates on remote application servers. A replica is a copy of the data that copies from one application server to another. The number of replicas that you configure affects the performance of your configuration. Smaller numbers of replicas result in better performance because the data does not have to be copied many times. However, if you create more replicas, you have more redundancy in your system. By configuring more replicas, your system becomes more tolerant to possible failures of application servers in the system because the data is backed up in several locations.

By having a single replica configuration defined, you can avoid a single point of failure in the system. However, if your system must be tolerant to more failure, introduce extra redundancy in the system. Increase the number of replicas that you create for any HTTP session that is replicated with DRS. Any replication domain that is used by dynamic cache must use a full group replica.

Session manager, dynamic cache, and stateful session beans are the three consumers of replication. A consumer is a component that uses the replication service. When you configure replication, the same types of consumers belong to the same replication domain. For example, if you are configuring both session manager and dynamic cache to use DRS to replicate objects, create separate replication domains for each consumer. Create one replication domain for all the session managers on all the application servers and one replication domain for the dynamic cache on all the application servers. The only exception to this rule is to create one replication domain if you are configuring replication for HTTP sessions and stateful session beans. Configuring one replication domain in this case ensures that the backup state information is located on the same backup application servers.

8.2.18 Service integration bus

A service integration bus supports applications that use message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

A service integration bus provides the following capabilities:

- Any application can exchange messages with any other application by using a destination to which one application sends and from which the other application receives.
- A message-producing application (a producer) can produce messages for a destination regardless of which messaging engine the producer uses to connect to the bus.
- A message-consuming application (a consumer) can consume messages from a destination (whenever that destination is available) regardless of which messaging engine the consumer uses to connect to the bus.

A service integration bus supports the following types of messaging:

- Sending messages synchronously (this requires the consuming application to be running and reachable).
- Sending messages asynchronously (possible whether the consuming application is running or not and whether or not the destination is reachable). Both point-to-point and publish/subscribe messaging are supported.
- Publishing events or other notifications. Notification messages can also be generated by the bus itself.

The bus appears to its applications as though it were a single logical entity, which means applications only need to connect to the bus and do not need to be aware of the bus topology. In many cases the knowledge of how to connect to the bus and which bus resources are defined are handled by a suitable API abstraction, such as the administered JMS connection factory and JMS destination objects

When a service integration bus has the role of providing the messaging system for JMS applications using the default messaging provider, it is sometimes referred to as a *messaging bus*.

Many scenarios only require relatively simple bus topologies, perhaps even just a single server. When integrating applications that have been deployed to multiple servers, it is often appropriate to add those servers as members of the same bus. However, servers do not have to be bus members to connect to a bus. In more complicated scenarios, multiple buses are configured, which might be interconnected to more complicated networks. An enterprise might deploy multiple interconnected buses for organizational reasons. For example, an enterprise with several autonomous departments might want to have separately administered buses in each location.

8.3 Security

Security usually requires a physical separation of the Web server from the application server processes, typically across one or more firewalls. A common configuration would be the use of two firewalls to create a DMZ between them. Information in the DMZ has some protection that is based on protocol and port filtering. A Web server intercepts the requests and forwards them to the corresponding application servers through the next firewall. The sensitive portions of the business logic and data resides behind the second firewall, which filters based on protocols, ports, IP addresses, and domains.

8.3.1 Concepts overview

This section describes some common security concepts necessary for the understanding of the subsequent sections of this chapter.

Authentication

Is the process of identifying who is requesting access to a resource. For the authentication process the server implements some challenge mechanism to gather unique information to identify the client. Secure authentication can be knowledge-based (user and password), key-based (physical keys, encryption keys), or biometric (fingerprints, retina scan, DNA, and so forth).

To increase security for the authentication mechanism itself, it will likely involve Lightweight Directory Access Protocol (LDAP) used with a Lightweight Third Party Authentication (LTPA) mechanism or a native operating system or a Custom Pluggable Registry.

Authorization

Authorization is the process of identifying what a client is allowed to do — what operations a client can do on a given resource. Resources can be Web pages, servlets, JSPs, and EJBs. There are two fundamental methods for authorization:

- Access Control List

  Each resource has associated with it a list of users and what each user can do with the resource (for example use, read, write, execute, delete, or create). Usually, an Access Control List specifies a set of roles that are allowed to use a particular resource and also designates the people that are allowed to fill these roles.

  When defining the Access Control List, each resource has a list of subjects (users, groups, roles, and so forth) with access to the resource.

- Capability List

  Associated with each user is a list of resources and the corresponding privileges held for the user. In this case, the holder is given the right to perform the operation on a particular resource.

Delegation

Delegation occurs when a client requests a method on server A. The method request results in a new invocation to another method of an object in server B. server A performs the authentication of the identity of the client and passes the request to server B. Server B assumes that the client identity has been verified by server A and responds to that request.

Secure Socket Layer

Secure Socket Layer (SSL) is the industry standard for data interchange encryption between clients and servers. SSL provides secure connections through:

- Communication privacy

  The data that passes through the connection is encrypted.

- Communication integrity

  The protocol includes a built-in integrity check.

- Authentication

  The server authenticates the client interchanging digital certificates.

A certificate is an encrypted, password protected file that includes:

- The name of the certificate holder
- The private key for encryption/decryption
- The verification of sender's public key
- The name of the Certification Authority
- Validity period for the certificate

A Certification Authority is an organization that issues certificates after verifying the requester's identity.

Single Signon

Single Signon (SSO) is the process where users provide their credentials (user ID, password, and token) just once within a session. These credentials are available to all enterprise applications for which SSO was enabled without prompting the user to re-enter user ID and password when switching from one application to another.

The following is a list with the requirements for enabling SSO using LTPA:

- All SSO participating servers must use the same user registry (for example, the LDAP server).
- All SSO participating servers must be in the same Domain Name System. (Cookies are issued with a domain name and will not work in a domain other than the one for which it was issued.)
- All URL requests must use domain names. No IP addresses or host names are allowed because these cause the cookie not to work properly.
- The browser must be configured to accept cookies.
- Server time and time zone must be correct. The SSO token expiration time is absolute.
- All servers participating in the SSO scenario must be configured to share LTPA keys.

Lightweight Third Party Authentication Protocol

LTPA is an authentication service that provides SSO and delegation. Authentication information is carried in LTPA tokens (cookies). It requires an LDAP directory service or custom registry. Operating system user registry is not supported.

Lightweight Directory Access Protocol

LDAP is a directory service, not a database. The information in the LDAP directory is descriptive, attribute based. LDAP users generally read the information much more than change it. The LDAP model is based on entries that are referred to as objects. Each entry consists of one or more attributes such as a name or address and a type. The types typically consist of mnemonic strings, such as *cn* for common name or *mail* for email address. Each directory entry has also a special attribute called objectClass. This attribute controls which attributes are required and allowed in each entry.

For more information about security concepts, see Chapter 17, “Planning for security” on page 487.

Public Key Infrastructure

A Public Key Infrastructure (PKI) represents a system of digital certificates, certificate authorities, registration authorities, a certificate management service, and X.500 directories.

A PKI verifies the identity and the authority of each party that is involved in an Internet transaction, either financial or operational, with requirements for identity verification. Examples of these transactions include confirming the origin of proposal bids, or the author of e-mail messages.

A PKI supports the use of certificate revocation lists (CRLs). A CRL is a list of revoked certificates. CRLs provide a more global method for authenticating client identity by certificate and can verify the validity of trusted CA certificates.

An X.500 directory server stores and retrieves CRLs and trusted CA certificates. The protocols used for storing and retrieving information from an X.500 directory server include Directory Access Protocol (DAP) and LDAP. The IBM HTTP Server supports LDAP.

You can distribute information about multiple directory servers over the Internet and intranets, enabling an organization to manage certificates, trust policy, and CRLs from either a central location, or in a distributed manner. This capability makes the trust policy more dynamic because you can add or delete trusted CAs from a network of secure servers, without having to reconfigure each of the servers.

8.4 Performance

Performance involves minimizing the response time for a given transaction. To reach this target development technics, server configuration, devices and setup for load balancing is crucial. This section highlights four major factors to consider for performance from the topology selection point of view. Performance is addressed in detail in Chapter 14, “Planning for performance, scalability, and high availability” on page 405.

8.4.1 Right hardware selection

The selection of the adequate hardware is key and can be achieved by having an early, very thorough, capacity planning. It should be performed by specialists that know the performance influence of each component in relation to the platform selected (number of processors, memory, DASD space, DASD arms, and others) because performance considerations and behaviors can vary from platform to platform.

8.4.2 Aligning to the best practices for development

Sometimes adding more and better hardware to support the implementation of a given application does not result in a performance improvement. The way the code is written, how well it flows along with the best practices for application development, can affect the performance of the application.

8.4.3 Proper application testing

You should perform a thorough application test phase before switching an application to production. In addition to the common tests for identifying flaws in the code, you should also include a set of stress test in this test phase. The idea is to run a load simulation of the actual workload that will exist in the production environment. The tests should be as realistic as possible (including similar hardware configurations when possible). They should reflect incremental load variations, different user behaviors, production size sample databases, and so forth to establish a baseline. Based on that baseline, further performance tuning might be required (at the application level) when you move to production. These tests also provide valuable information about how to tune the application server. Combining the results from the tests, you should be able to extrapolate those results to the production environment to get a realistic projection on how the application should behave when moved to production.

8.4.4 Scaling

Scaling represents the ability to create more application server processes to serve the user requests. Multiple machines can be configured to add processing power, improve security, maximize availability, and balance workloads. There are two options for scaling: vertical or horizontal. Which you use depends on where and how the scaling is taking place.

- Vertical scaling

  Involves creating additional application server processes on a single physical machine, providing application server failover as well as load balancing across multiple JVM (application server processes).

- Horizontal scaling

  Involves creating additional application server processes on multiple physical machines to take advantage of the additional processing capacity that is available on each machine, which also provides hardware failover support.

The components that provide functions for configuring scalability include:

- WebSphere Application Server cluster support

  Clusters allow the creation of a logical group of application servers that run the same application(s). Members of a cluster can be located on the same machine (vertical scaling) and across multiple machines (horizontal scaling).

  The use of application server clusters can improve the performance of a server, simplify its administration, and enable the use of workload management. However, an administrator needs to assess whether the cost of implementing a clustered configuration (network traffic, performance) outweighs the benefits of a clustered environment.

- WebSphere workload management

  The Workload Manager (WLM) distributes all the incoming requests from the clients among the clustered application servers. WLM enables both load balancing and failover, improving the reliability and scalability of WebSphere applications.

- IP sprayer

  The IP sprayer transparently redirects all incoming HTTP requests from Web clients to a group of Web servers. Although the clients behave as though they are communicating directly with a one specific Web server, the IP sprayer is actually intercepting all those requests and distributing them among all the available Web servers in the group. IP sprayers (such as the Load Balancer component of Edge Components or Cisco Local Director) can provide scalability, load balancing, and failover for Web servers.

8.5 Throughput

Throughput is also related to performance but in the sense of the volume being served. This volume can be measured in the number of bytes that are transferred or the number of transactions that are completed in a given period of time. For example, if a site can serve 10 requests simultaneously and each request takes one second to process, this site can have a potential throughput of 10 requests per second.

WebSphere Application Server Network Deployment provides clusters, which logically group a number of application servers that run the same application.

The application servers are added through vertical and horizontal scaling. These configurations provide improvements for both performance (response time) and throughput (how many).

8.6 Availability

To avoid a single point of failure and to maximize system availability, the topology should have some degree of process redundancy. High-availability topologies typically involve horizontal scaling across multiple machines. Vertical scaling can improve availability by creating multiple processes, but the machine itself becomes a point of failure.

In addition, an IP sprayer can direct Web client requests to the next available Web server, bypassing any server that is offline. The IP sprayer server can be configured with a cascade backup (standby) that comes online in the event that the primary server fails. This configuration eliminates single point of failure on the IP sprayer.

Improved availability is one of the key benefits of scaling to multiple machines. WebSphere Application Server Network Deployment provides tools that let you manage availability by distributing critical functionality across multiple machines. Applications that are hosted on multiple machines generally have less down time and are able to service client requests more consistently.

This section discusses the commonly used techniques that you can combine to take advantage of the best features of each topology and create a highly available system.

8.6.1 Hardware and process redundancy

Eliminate the single points of failure in a system by including hardware and process redundancy using the following techniques:

- Use horizontal scaling to distribute application servers (and applications) across multiple physical machines. If a hardware or process failure occurs, clustered application servers are available to handle client requests. Additional Web servers and IP sprayers can also be included in horizontal scaling to provide higher availability.
- Use backup servers for databases, Web servers, IP sprayers, and other important resources, ensuring that they remain available if a hardware or process failure occurs.
- Keep the servers (physical machines) within the cluster sprayed in different locations to prevent site related problems.

8.6.2 Process isolation

Provide process isolation so that failing servers do not impact the remaining healthy servers of a configuration negatively. The following configurations provide some degree of process isolation:

- Deploy the Web server onto a different machine from the application servers. This configuration ensures that problems with the application servers do not affect the Web server and vice versa.
- Use horizontal scaling, which physically separates application server processes onto different machines.

8.6.3 Load balancing

Use load-balancing techniques to make sure that individual servers are not overwhelmed with client requests while other servers are idle. These techniques include:

- Use of an IP sprayer to distribute requests across the Web servers in the configuration.
- Direct requests from high-traffic URLs to more powerful servers. The Edge Components included with WebSphere Application Server Network Deployment provide these features.

8.6.4 Failover support

The site must be able to continue processing client request even if some of the servers are offline. Some ways to provide failover support include:

- Use horizontal scaling with workload management to take advantage of its failover support.
- Use of an IP sprayer to distribute requests across the Web servers in the configuration.
- Use HTTP transport to distribute client requests among application servers.

8.6.5 Hardware-based high availability

The WebSphere Application Server high availability framework provides integration into an environment that might be using other high availability frameworks, such as HACMP, to manage resources that do not use WebSphere.

8.7 Maintainability

The topology affects the ease with which system hardware and software can be updated. For instance, using horizontal scaling can make a system easier to maintain because individual machines can be taken offline without interrupting other machines running the application, thus providing business continuity. When deciding how much vertical and horizontal scaling to use in a topology, consider needs for hardware upgrades (for example, adding or upgrading processors and memory).

8.8 Session management

Multimachine scaling techniques rely on using multiple copies of an application server. Multiple consecutive requests from various clients can be serviced by different servers. If each client request is completely independent of every other client request, it does not matter whether consecutive requests are processed on the same server. However, in practice, client requests are not independent. A client often makes a request, waits for the result, then makes one or more subsequent requests that depend on the results received from the earlier requests. This sequence of operations on behalf of a client falls into two categories:

- Stateless

  A server processes requests based solely on information provided with each request and does not reply on information from earlier requests. In other words, the server does not need to maintain state information between requests.

- Stateful

  A server processes requests based on both the information that is provided with each request and information that is stored from earlier requests. In other words, the server needs to access and maintain state information that is generated during the processing of an earlier request.

For stateless interactions, it does not matter whether different requests are processed by different servers.

However, for stateful interactions, the server that processes a request needs access to the state information necessary to service that request. Either the same server can process all requests that are associated with the same state information or the state information can be shared by all servers that require it. In the latter case, accessing the shared state information from the same server minimizes the processing overhead associated with accessing the shared state information from multiple servers.

The load distribution facilities in WebSphere Application Server use several different techniques for maintaining state information between client requests:

- Session affinity, where the load distribution facility recognizes the existence of a client session and attempts to direct all requests within that session to the same server.
- Transaction affinity, where the load distribution facility recognizes the existence of a transaction and attempts to direct all requests within the scope of that transaction to the same server.
- Server affinity, where the load distribution facility recognizes that although multiple servers might be acceptable for a given client request, a particular server is best suited for processing that request.

The WebSphere Application Server Session Manager, which is part of each application server, stores client session information and takes session affinity and server affinity into account when directing client requests to the Cluster Members of an application server. The workload management service takes server affinity and transaction affinity into account when directing client requests among the cluster members of an application server.

Sessions

A session is a series of requests to a servlet that originate from the same user at the same browser. Sessions allow applications running in a Web container to keep track of individual users. For example, a servlet might use sessions to provide shopping carts to online shoppers.

Suppose the servlet is designed to record the items each shopper indicates he or she wants to purchase from the Web site. It is important that the servlet be able to associate incoming requests with particular shoppers. Otherwise, the servlet might mistakenly add Shopper_1's choices to the cart of Shopper_2.

A servlet distinguishes users by their unique session IDs. The session ID arrives with each request. If the user's browser is cookie-enabled, the session ID is stored as a cookie. As an alternative, the session ID can be conveyed to the servlet by URL rewriting, in which the session ID is appended to the URL of the servlet or JSP file from which the user is making requests. For requests over HTTPS or SSL, an alternative is to use SSL information to identify the session.

Session management support

WebSphere Application Server provides facilities, grouped under the heading Session Management, that support the javax.servlet.http.HttpSession interface that is described in the Servlet API specification. In accordance with the Servlet 2.3 API specification, the Session Management facility supports session scoping by Web module. Only servlets in the same Web module can access the data that is associated with a particular session. Multiple requests from the same browser, each specifying a unique Web application, result in multiple sessions with a shared session ID. You can invalidate any of the sessions that share a session ID without affecting the other sessions.

You can configure a session timeout for each Web application. A Web application timeout value of 0 (the default value) means that the invalidation timeout value from the Session Management facility is used. When an HTTP client interacts with a servlet, the state information associated with a series of client requests is represented as an HTTP session and identified by a session ID. Session Management is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request through the use of cookies or URL rewriting techniques.

Session Management can store session-related information in several ways:

- In application server memory (the default). This information cannot be shared with other application servers.
- In a database. This storage option is known as *database persistent sessions*.
- In another WebSphere Application Server instance. This storage option is known as *memory-to-memory sessions*.

The last two options are referred to as *distributed sessions*. Distributed sessions are essential for using HTTP sessions for failover facility. When an application server receives a request associated with a session ID that it currently does not have in memory, it can obtain the required session state by accessing the external store (database or memory-to-memory). If distributed session support is not enabled, an application server cannot access session information for HTTP requests that are sent to servers other than the one where the session was originally created. Session Management implements caching optimizations to minimize the overhead of accessing the external store, especially when consecutive requests are routed to the same application server.

Storing session states in an external store also provides a degree of fault tolerance. If an application server goes offline, the state of its current sessions is still available in the external store. This availability enables other application servers to continue processing subsequent client requests associated with that session.

Saving session states to an external location does not completely guarantee their preservation in case of a server failure. For example, if a server fails while it is modifying the state of a session, some information is lost and subsequent processing using that session can be affected. However, this situation represents a very small period of time when there is a risk of losing session information.

The drawback to saving session states in an external store is that accessing the session state in an external location can use valuable system resources. Session Management can improve system performance by caching the session data at the server level. Multiple consecutive requests that are directed to the same server can find the required state data in the cache, reducing the number of times that the actual session state is accessed in external store and consequently reducing the overhead associated with external location access.

Distributed sessions

WebSphere Application Server provides the following session mechanisms in a distributed environment:

- Database Session persistence, where sessions are stored in the database specified.
- Memory-to-memory Session replication, where sessions are stored in one or more specified WebSphere Application Server instances.

When a session contains attributes that implement HttpSessionActivationListener, notification occurs anytime the session is activated (that is, session is read to the memory cache) or passivated (that is, session leaves the memory cache). Passivation can occur because of a server shutdown or when the session memory cache is full and an older session is removed from the memory cache to make room for a newer session. It is not guaranteed that a session is passivated in one application server prior to being activated in another. You can find more information about managing HTTP sessions at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

Unless only one application server is used, or the application is completely stateless. Maintaining a session state between Web client requests is a factor in determining the chosen topology. When session state is an issue, you need to consider the right configuration of the IP sprayer, if any.

8.9 Topology selection summary

The following tables provide a summary of possible considerations for topology selection as discussed in this chapter. These tables list the requirements (such as availability, performance, security, maintainability, and session persistence) and the possible solution(s) for the Web server, the application server(s), and the database server.

Table 8-1 Topology selection based on availability requirements

Requirement = Availability	**Solution/Topology**
Web server	Either Edge Components Load Balancer (with backup) or HA solution, based on additional requirements
Application server	► Horizontal scaling (process and hardware redundancy) ► Vertical scaling (process redundancy) ► A combination of both
Database server	HA software solution

Table 8-2 Topology selection based on performance requirements

Requirement = Performance/Throughput	**Solution/Topology**
Web server	► Multiple Web servers in conjunction with Edge Components Load Balancer ► Caching proxy ► Dynamic caching with AFPA or ESI external caching
Application server	► Clustering (in most cases horizontal) ► Dynamic caching ► Serving static content from Web server to offload application server
Database server	Separate database server

Table 8-3 Topology selection based on security requirements

Requirement = Security	Solution/Topology
Web server	Separate the Web server into a DMZ, either on LPAR or separate system
Application server	Separate components into a DMZ, implement security (for example, LDAP)
Database server	Not covered

Table 8-4 Topology selection based on maintainability requirements

Requirement = Maintainability	Solution/Topology
Web server	Single machine environment is the easiest option to maintain. It can be combined with horizontal scaling.
Application server	
Database server	

Table 8-5 Topology selection based on session affinity requirements

Requirement = Session persistence	Solution/Topology
Web server	Session affinity is handled by the WebSphere plug-in
Application server	► Session database (possible SPOF if not HA) ► Memory-to-memory replication
Database server	Not covered

9

Topologies

This chapter covers some common topologies that are used in WebSphere Application Server implementations. It addresses topologies that are more simple to those that are more complex by describing the relationship between WebSphere Application Server components and their role in the solution to implement. This chapter contains the following sections:

- Single machine topology (stand-alone server)
- Reverse proxy topology
- Cell topologies
- Web server topology in a Network Deployment cell
- Mixed node versions in a cell topology
- Clustered topologies
- Mixed node versions in a clustered topology
- Vertical scaling topology
- Horizontal scaling topology
- Horizontal scaling with IP sprayer topology
- Topology with redundancy of several components

9.1 Single machine topology (stand-alone server)

A single machine topology refers to the installation of WebSphere Application Server on one single (physical) machine or Logical Partition (LPAR).

9.1.1 Multiple installs

You can install WebSphere Application Server V6 several times on the same machine in different directories. Those installations are independent from each other. This configuration facilitates fix management. If a fix is applied on a particular installation, it only affects that specific WebSphere Application Server installation leaving unaffected the remaining installations on that machine.

9.1.2 Single install, multiple WebSphere profiles

When you install WebSphere Application Server V6 (single installation), you can still have multiple WebSphere profiles. This concept was introduced in the previous version of WebSphere Application Server as WebSphere Instances. In this case, all profiles share the same product binaries. When fixes are installed, they affect all profiles. Each profile has its own user data.

Figure 9-1 shows the difference between multiple installs and multiple WebSphere profiles.

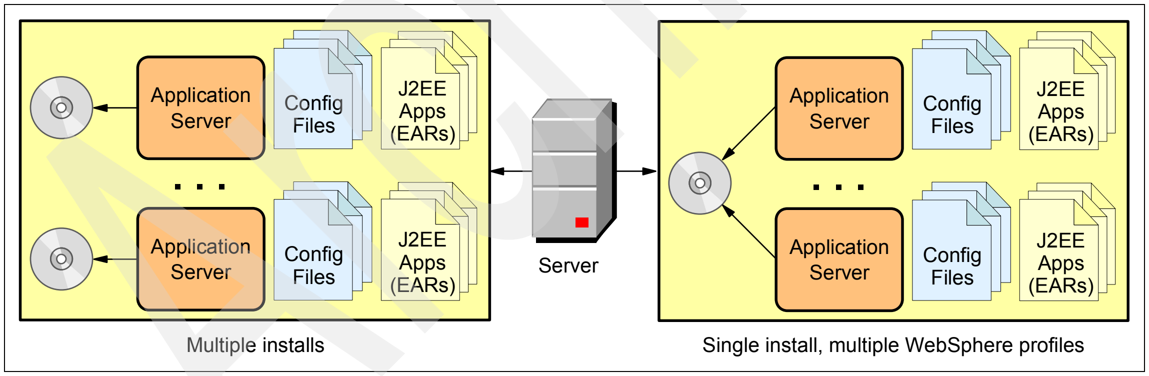


Figure 9-1 Stand-alone server installation options

Note: There is no limit for multiple installs or multiple profiles. The real limitation is ruled by the hardware capacity.

9.1.3 Single machine, single node, one application server

The starting scenario for this discussion is the configuration where all components reside on the same machine as shown in Figure 9-2. The Web server routes requests, as appropriate, to the WebSphere Application Server on the same machine for processing.

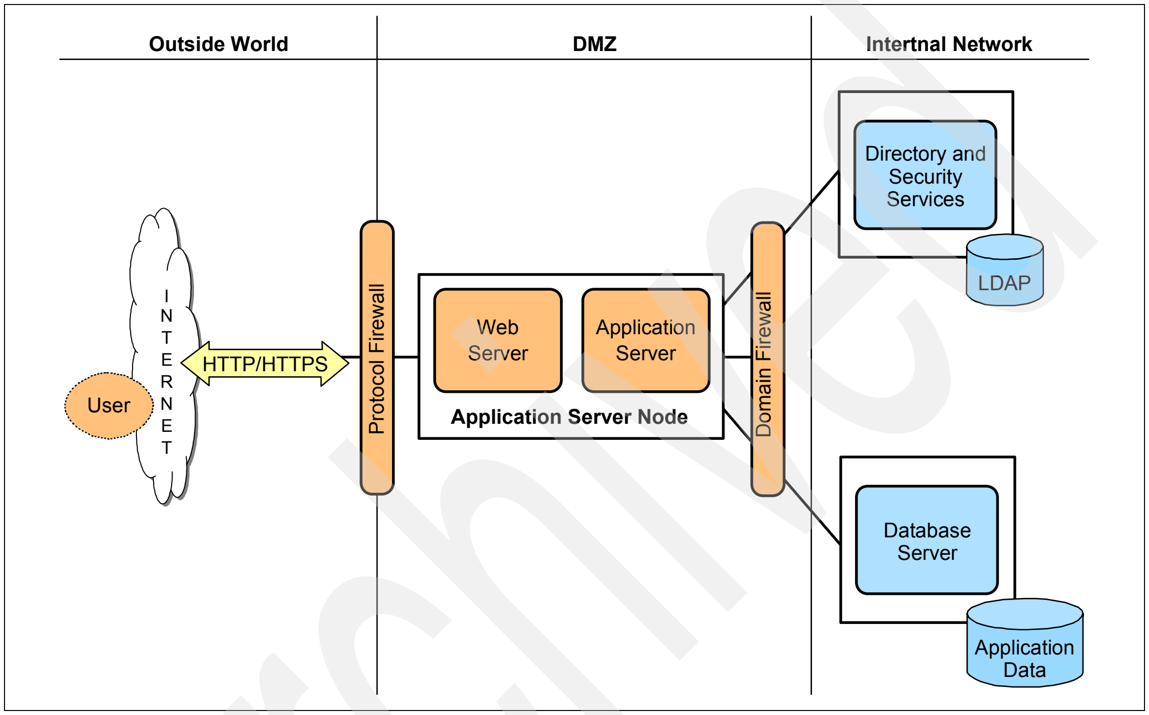


Figure 9-2 Stand-alone topology option

In this configuration, there is one application server node that contains one Web server and one application server running applications. Variations of this topology are possible. This configuration also includes two firewalls to create a demilitarized zone (DMZ) and a directory server to handle the user authentication. You can create a much simpler configuration, without firewalls and a security implementation for development or testing purposes.

Advantages

Some good reasons to use a single machine topology are:

- Maintainability

  A single machine topology is easy to install and maintain. This configuration is most suitable as a startup configuration to evaluate and test the basic functionality of WebSphere Application Server and related components. The installation is automated by tools that are supplied with the WebSphere distribution. This configuration is also the easiest to administer.

- Low cost

  Because the components are located in the same machine, the cost of the hardware is reduced and also the cost of maintenance.

Disadvantages

You should also consider the following disadvantages when you use a single machine topology:

- Performance

  One disadvantage of a single machine topology is components interdependence and competition for resources. All components compete for the shared resources (processor, memory, network, I/O, and so on). Because components influence each other, bottlenecks or ill-behaved components can be difficult to identify and remediate.

- Security

  With a single machine topology, no isolation between Web server and application server is available. If firewalls and security are not configured, there is no explicit layer of isolation between the components.

- Availability

  This configuration is a single point of failure. If the hardware or the Web server or the application server fails, the entire site is not usable.

9.1.4 Web server separated

When compared to a configuration where the application server and the Web server are collocated on a single physical server, separation of the application server and the Web server can be used to provide improvement in security, performance, throughput, availability, and maintainability. Figure 9-3 illustrates this topology.

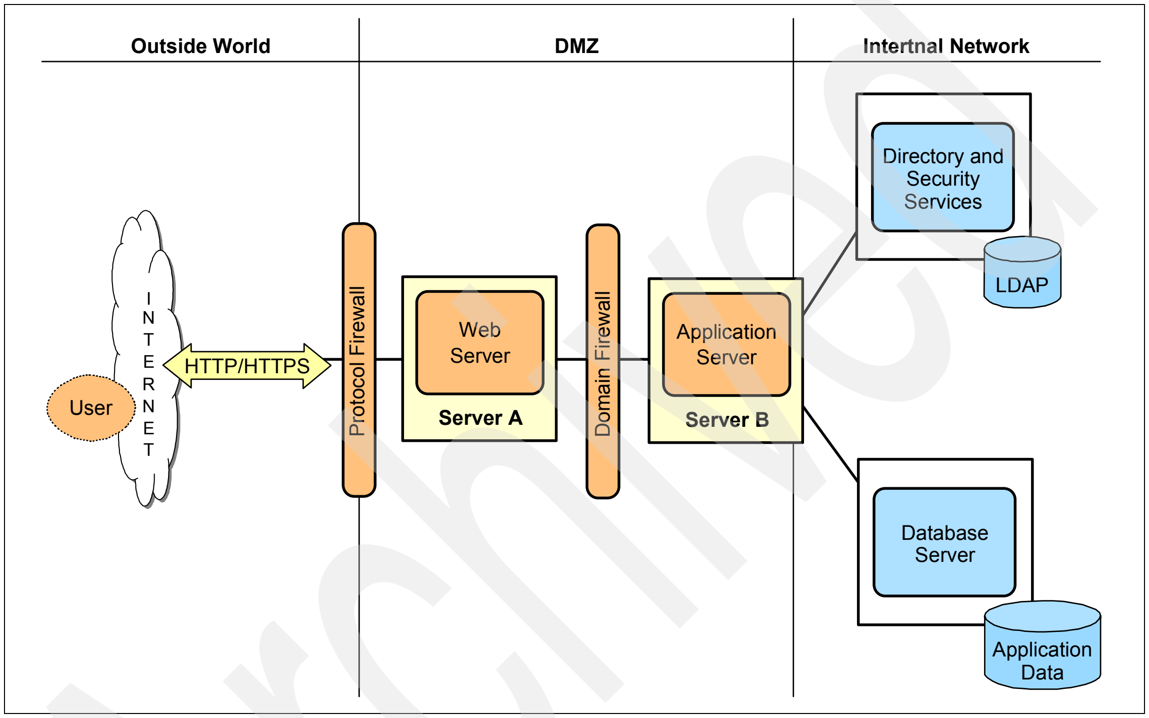


Figure 9-3 Web server separation

The Web server plug-in allows the Web server to route requests to the application server when they are physically separated. It uses an XML configuration file (plugin-cfg.xml) that contains settings that describe how to handle and pass on requests to the WebSphere application server(s). Be aware that in this case the plugin-cfg.xml configuration file is generated on the machine where the application server is installed so it has to be moved, each time it is regenerated, from the machine where the application server resides to the machine where the Web server and the plug-in module are installed.

A failure on the Web server could be bypassed pointing the Domain Name System to the machine where WebSphere Application Server is installed. This way, the embedded WebSphere Application Server Web server can replace the Web server (with limited throughput) while the problem is solved.

Advantages

Some reasons to separate the Web server from the application server are:

- Performance
  - Size and configure servers appropriately to each task.

    By installing components (Web server and application server) on separate machines, each machine can be sized and configured to optimize the performance of each component.
  - Remove resource contention.

    By installing the Web server on a separate machine from the WebSphere Application Server machine, a high load of static requests will not affect the resources (processor, memory, and disk) available to WebSphere, and therefore does not affect its ability to service dynamic requests. The same applies when the Web server serves dynamic content using other technologies, such as CGI and so on.
- Maintainability

  Web server separation provides component independence. Server components can be reconfigured, or even replaced, without affecting the installation of other components on separate machines.
- Security

  This configuration provides separate Web server and WebSphere interfaces. To protect the application servers from unauthorized outside access, the separation of the Web server from the application server is often used with firewalls to create a secure DMZ surrounding the Web server. Isolating the Web server on a DMZ protects the application logic and data by restricting the access from the public Web site to the servers and databases where this valuable information is stored. Desirable topologies should have neither databases nor servers that directly access databases, in the DMZ. WebSphere Application Server stores the configuration data as XML files. Furthermore, an application installed on WebSphere usually needs to access a database. For this reason, it is not a recommended solution to run WebSphere Application Server in the DMZ.

Disadvantages

You should consider the following disadvantages when you separate the Web server from the application server:

- Maintainability

  This configuration is complex. The configuration file (plugin-cfg.xml) is generated on the WebSphere Application Server machine and must be copied to the Web server machine.

- Performance

  Separating the Web server from the application server can limit performance of network access. Depending upon the network capacity and remoteness of the Web server, the network response time for communications between WebSphere Application Server and the Web server can limit the application response time. To prevent this you should insure that you have an adequate network bandwidth between the Web server and the application server. When collocated on the same server, network response is usually not an issue, but other resource constraints such as memory and CPU can limit performance.

- Security considerations:

  This configuration uses encrypted transport. The HTTP plug-in allows encryption of the link between the Web server and the application server, using HTTP over Secure Socket Layer (HTTPS) data encryption. This reduces the risk that attackers would be able to obtain secure information by “sniffing” packets sent between the Web server and application server. A performance penalty usually accompanies such encryption. The HTTP plug-in is suitable for environments that require all network communication to be encrypted, even in a DMZ.

  It is recommended that this connection is configured so that the HTTP plug-in and Web container must mutually authenticate each other using public-key infrastructure. This prevents unauthorized access to the Web container.

9.1.5 Software requirements

Table 9-1 indicates the minimum software components that you need to install to configure a single machine topology.There are considerations about installation and configuration that are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-1 Minimum software required

Product / Component Name	Needed
WebSphere Application Server - Express or WebSphere Application Server	Yes
WebSphere Application Server Network Deployment	No
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.2 Reverse proxy topology

Reverse proxy (or IP-forwarding) topology uses a reverse proxy server, such as the one in Edge Components, to receive incoming HTTP requests and to forward them to a Web server. The Web server in turn forwards the requests to the application servers which do the actual processing. The reverse proxy returns requests to the client, effectively hiding the originating Web server. Figure 9-4 on page 255 illustrates this configuration.

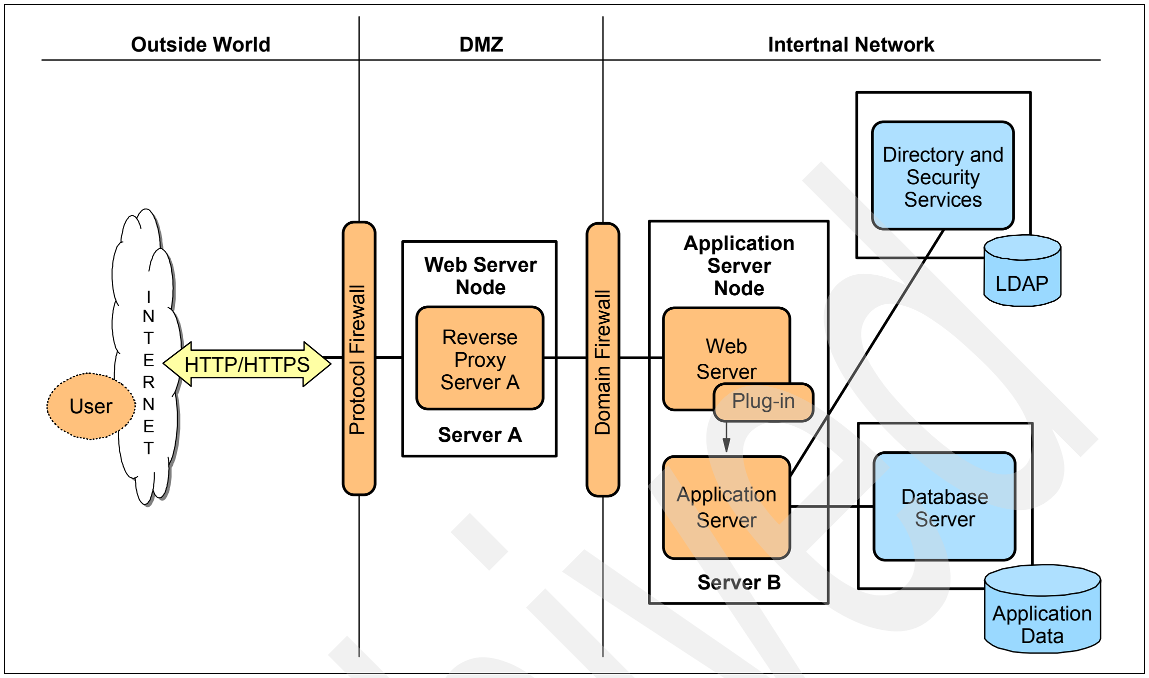


Figure 9-4 Topology using a reverse proxy

In this example, a reverse proxy resides in a DMZ between the outer and inner firewalls. It listens on the HTTP port (typically port 80) for HTTP requests. The reverse proxy then forwards those requests to the Web server that resides on the same machine as the application server. After the requests are fulfilled, they are returned through the reverse proxy to the Web client who made those requests, hiding the Web server.

Reverse proxy servers are typically used in DMZ configurations to provide additional security between the public Internet and the Web servers (and application servers) servicing requests.

Reverse proxy configurations support high-performance DMZ solutions that require as few open ports in the firewall as possible. For the reverse proxy in the DMZ to access the Web server behind the domain firewall requires as few as one port open in the firewall (potentially two ports if using SSL).

Advantages

Advantages of using a reverse proxy server in a DMZ configuration include:

- This is a well-known and tested configuration. It is, therefore, easy to implement.
- It is a reliable and fast-performing solution.
- It eliminates protocol switching by using the HTTP protocol for all forwarded requests.
- It has no effect on the configuration and maintenance of a WebSphere application.

Disadvantages

Disadvantages of using a reverse proxy server in a DMZ configuration include:

- It requires more hardware and software than similar topologies that do not include a reverse proxy server, making it more complicated to configure and maintain.
- The reverse proxy does not participate in WebSphere workload management.
- It cannot be used in environments where security policies prohibit the same port or protocol being used for inbound and outbound traffic across a firewall.

9.2.1 Software requirements

Table 9-2 indicates the minimum software components that you need to install to configure the reverse proxy topology. There are considerations about installation and configuration that are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-2 Minimum software required

Product / Component Name	**Needed**
WebSphere Application Server - Express or WebSphere Application Server	Yes
WebSphere Application Server Network Deployment	No
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	Yes

9.3 Cell topologies

A cell is a network of multiple nodes. It provides a single logical administration domain. It contains only one Deployment Manager. Deployment Manager and nodes can be on the same or a different machine or LPAR. Figure 9-5 illustrates the cell concept.

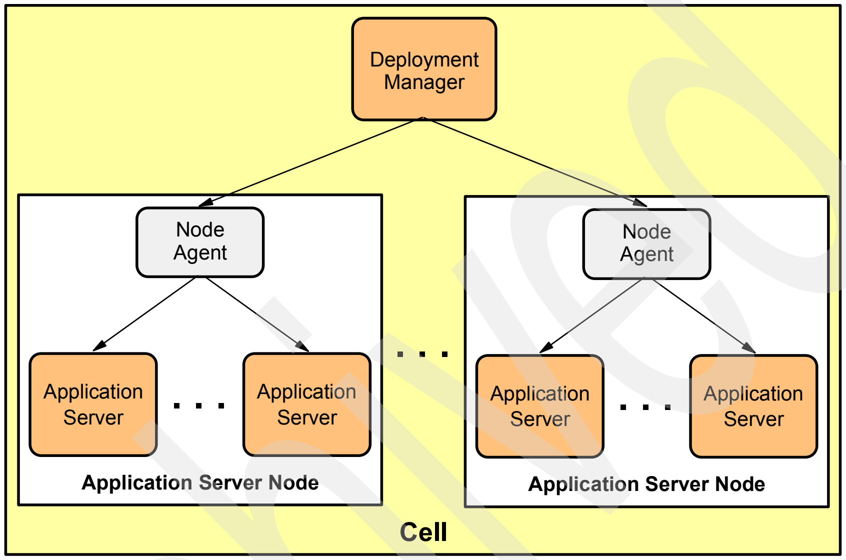


Figure 9-5 Cell topology option

For the creation of a cell, you must have WebSphere Application Server Network Deployment V6 install. The Deployment Manager manages one or more nodes in a distributed topology (cell). It communicates with the Node Agent for configuration and operational management of the application servers within that node.

Deployment Manager install options

You can install WebSphere Application Server Network Deployment V6 several times on the same machine (multiple installs), each one for administering different cells. Also, it is possible to install WebSphere Application Server Network Deployment V6 once and then create multiple profiles so that each profile is used to administer a different cell.

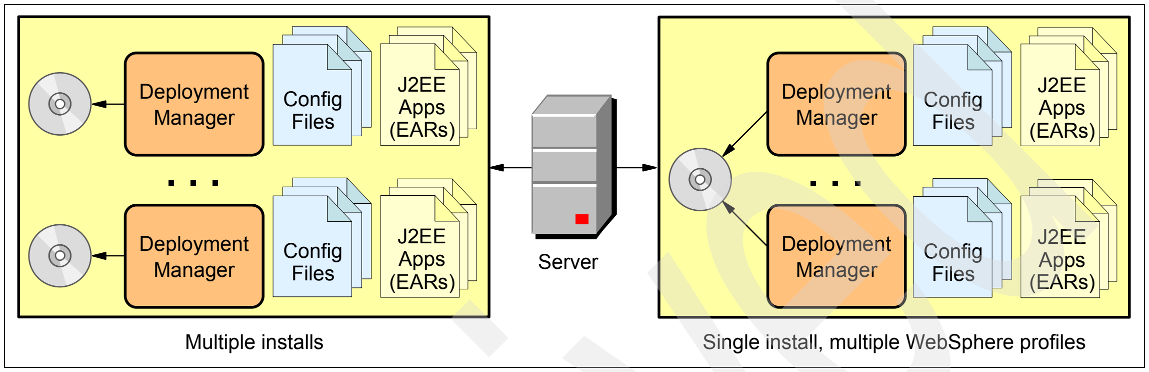


Figure 9-6 Deployment Manager, install options

Cell configuration flexibility

WebSphere Application Server Network Deployment V6 allows a variety of configurations. Several components can coexist on the same machine or be separated among different machines to satisfy the criteria of topology selection. Figure 9-7 on page 259 depicts a topology where there are two cells. Each cell is managed by only one Deployment Manager.

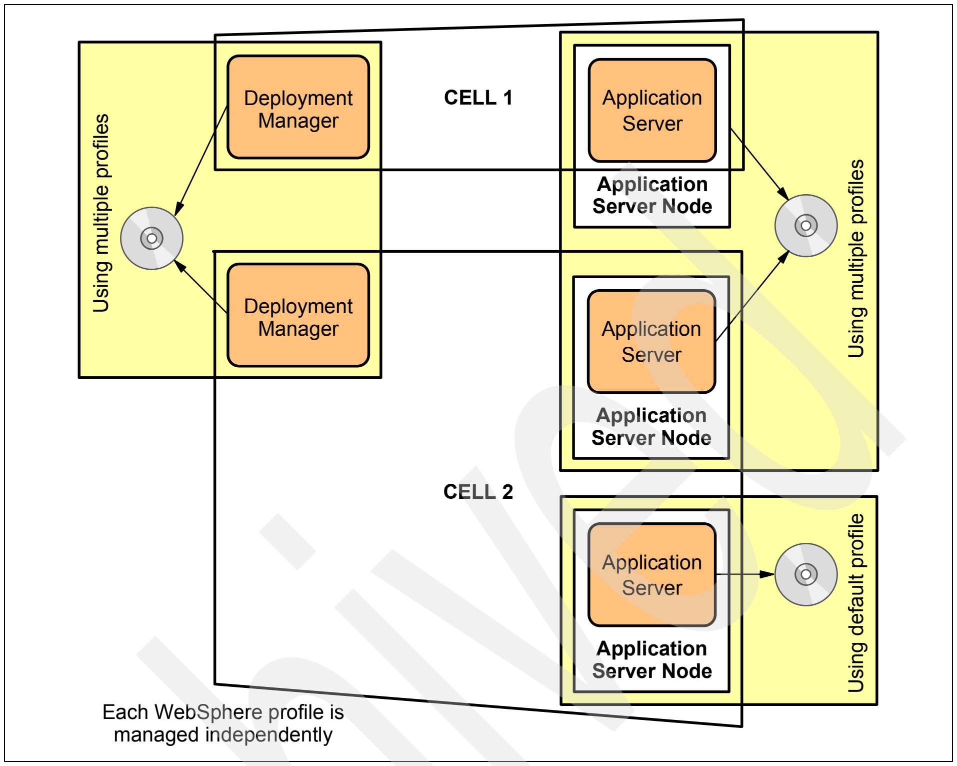


Figure 9-7 Cell configuration flexibility

Although the Deployment Managers reside in the same machine (single installation), they belong to different WebSphere profiles which provides independency from each other. The same concept applies to the application server nodes. They use a single install, but in one case, with multiple WebSphere profiles and in the other, a single install with a single WebSphere profile.

9.3.1 Deployment Manager and each node in separate machines

Deployment Manager can be installed on one machine (Server A) and each node on a different machine (Server B and Server C). The Deployment Manager or any other node can be on a different platform each.

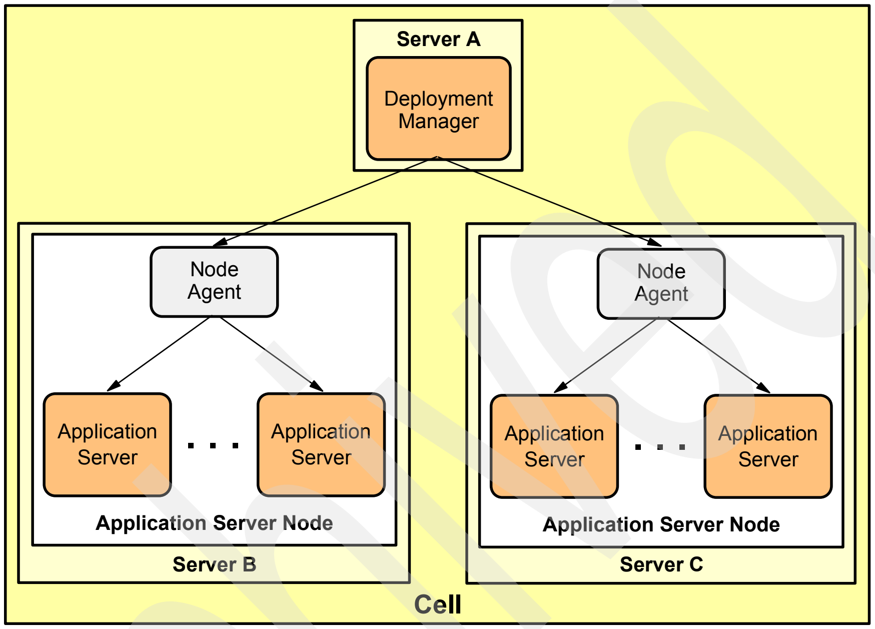


Figure 9-8 One cell, multiple nodes, Deployment Manager separated

9.3.2 Deployment Manager and all nodes in one single machine

If there is enough hardware capacity, Deployment Manager and one or more nodes can be configured on the same machine or LPAR as depicted in Figure 9-9.

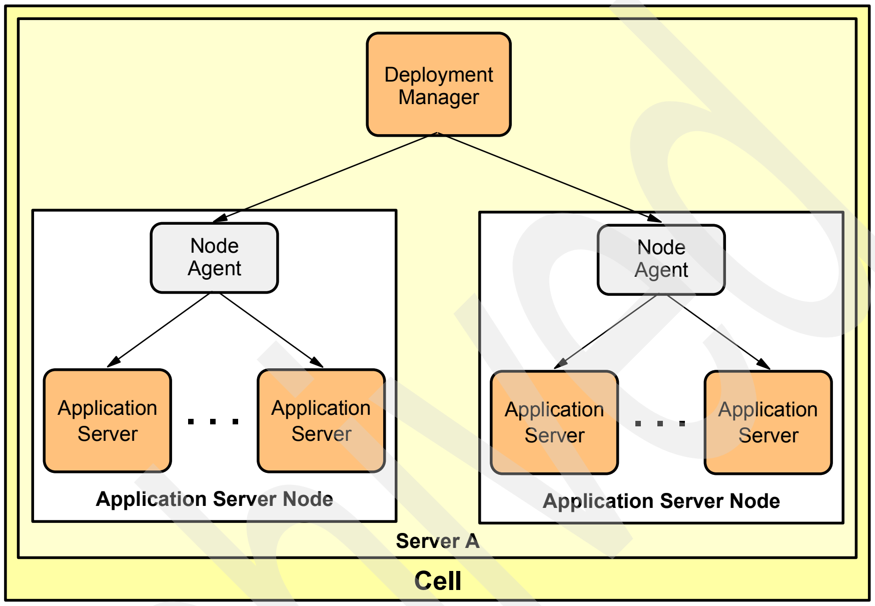


Figure 9-9 One cell, Deployment Manager and nodes on the same machine

9.3.3 Multiple nodes on the same machine or LPAR

Deployment Manager can reside on one machine and the Managed Nodes can be all on another server. Introduced in this release of WebSphere Application Server, more than one node can exist on the same host. Figure 9-10 illustrates this configuration.

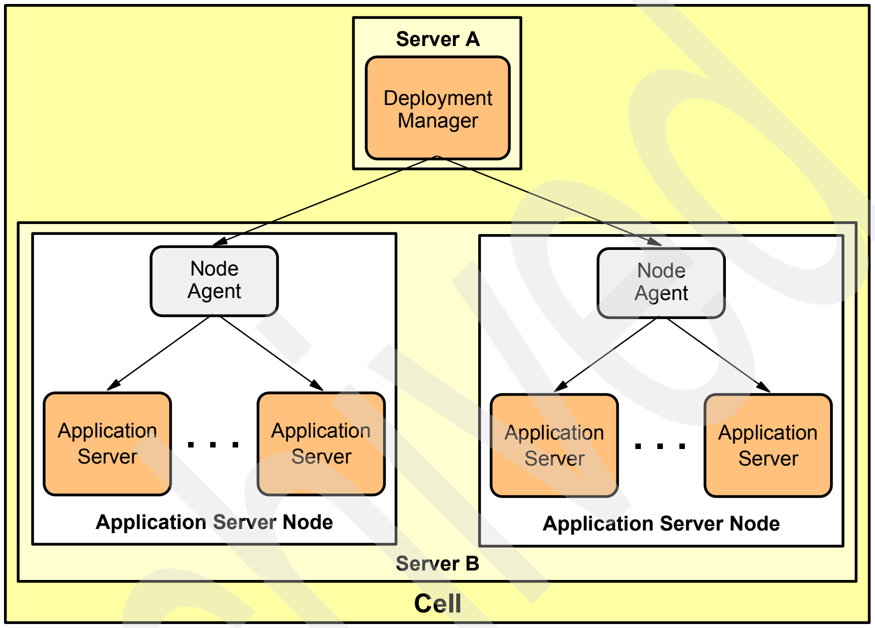


Figure 9-10 One cell, several nodes, Deployment Manager on another machine

9.3.4 Deployment Manager and one node on the same machine

One or more nodes can be placed on the same machine where Deployment Manager is installed while some other nodes can be on a separated machine. Figure 9-11 illustrates two servers within a cell. One of those servers is housing both an application server node and a Deployment Manager which manages the cell.

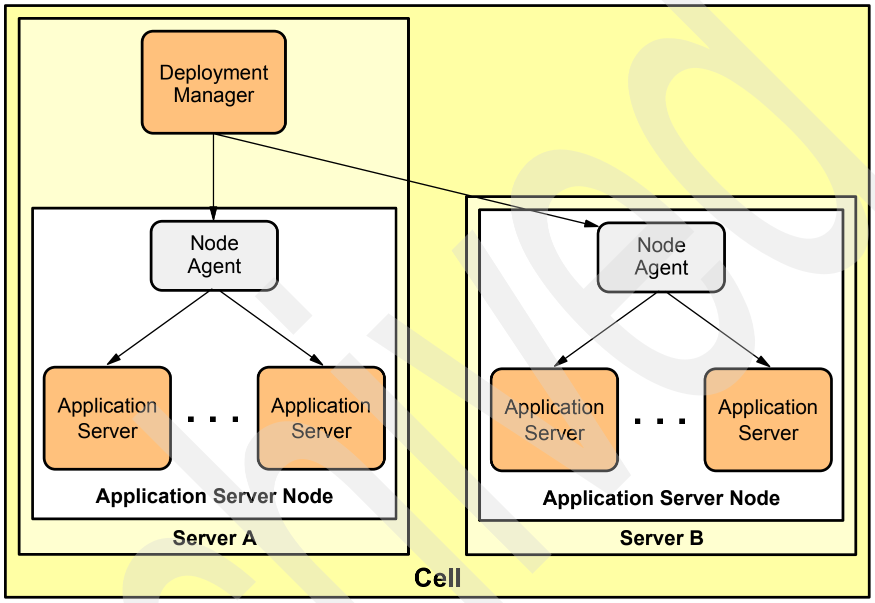


Figure 9-11 One cell, Deployment Manager, node on same machine, node separated

9.3.5 Parts of multiple cells share a physical machine

Several nodes can be created on a machine. Those nodes can be part of the same cell managed by a local Deployment Manager or part of a different cell managed by a remote Deployment Manager installed on a separated machine. Additionally these cells can reside entirely within one server or be sprayed among two or more servers as shown in Figure 9-12.

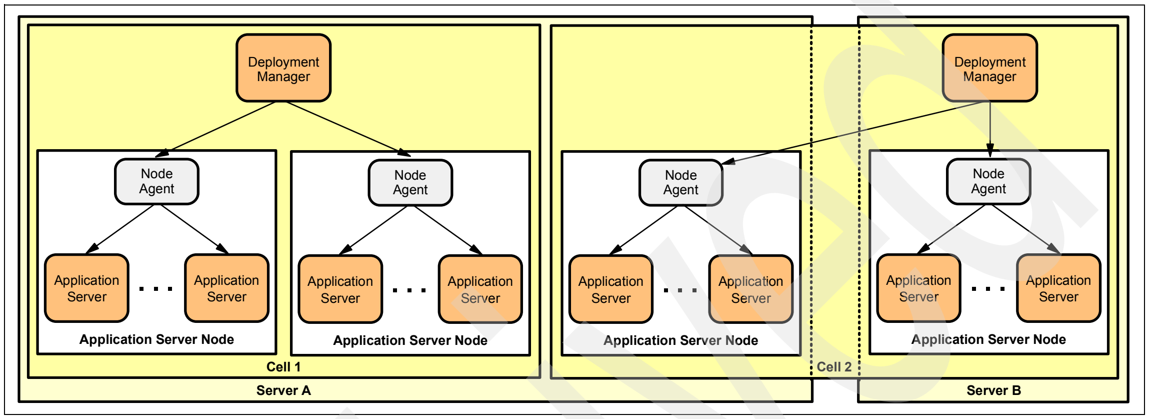


Figure 9-12 Multiple independent cells can share a physical machine or LPAR

9.3.6 Software requirements

Table 9-3 indicates the minimum software components that you need to install to configure the cell topologies. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-3 Minimum software required

Product / Component Name	**Needed**
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.4 Web server topology in a Network Deployment cell

In WebSphere Application Server V6, a Web server can be defined in a cell as a Web server node. This allows the association of applications to one or more Web servers. A custom plug-in configuration file can be generated for specific Web servers. The subsequent sections cover in detail some options for this topology. For information about the plugin configuration, see 10.3.5, “Planning for Web server plug-in installation” on page 290.

9.4.1 Web server Managed Node

Managed Node means that the Web server is managed by the Deployment Manager. This configuration provides the ability to start and stop the Web server from WebSphere Application Server Network Deployment V6 console and automatically push the plug-in configuration file to the Web server. It requires a Node Agent to be installed on the Web server machine.

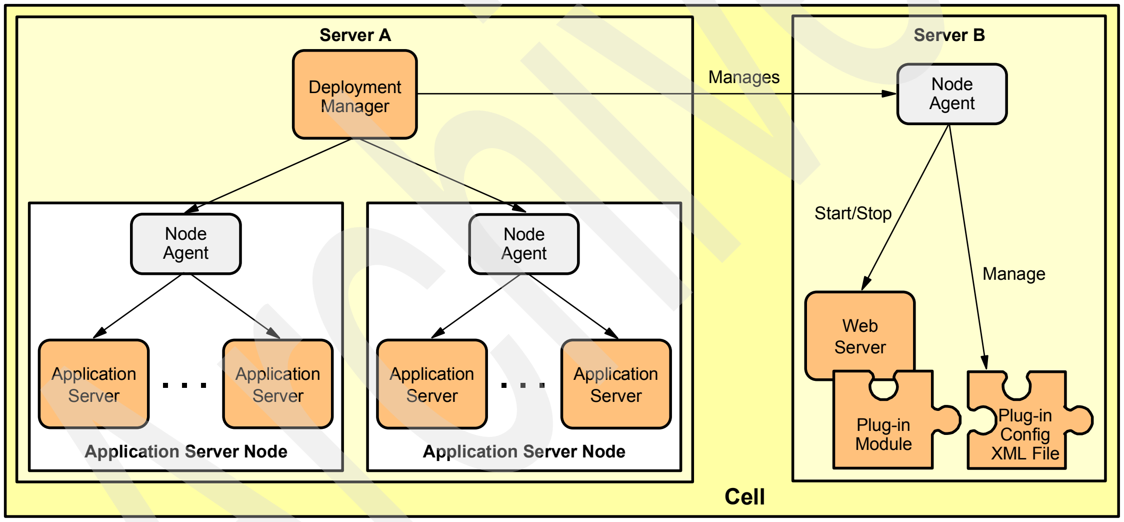


Figure 9-13 Web server Managed Node

9.4.2 Web server Unmanaged Node

This is the typical configuration that was well known before the concept of Managed Node. It is a common option for Web servers installed outside a firewall where no Node Agent is installed. The use of this topology requires that each time the plug-in configuration file is regenerated, it has to be copied from the machine where WebSphere Application Server is installed to the machine where the Web server is running.

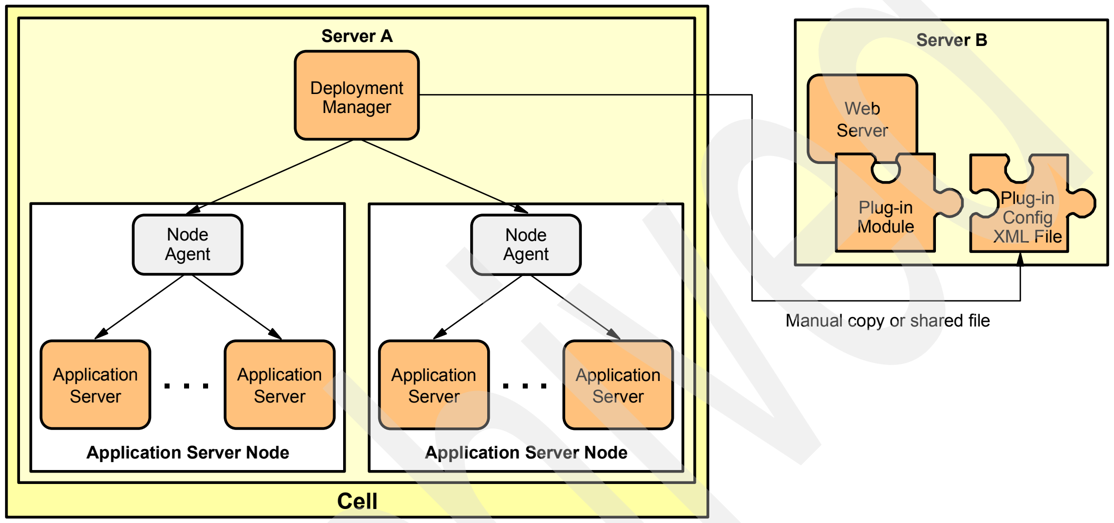


Figure 9-14 Web server Unmanaged Node

9.4.3 IBM HTTP Server as Unmanaged Node (special case)

If the Web server is the IBM HTTP Server, then it can be installed on a remote machine without installing a Node Agent. You can administer the IBM HTTP Server through the Deployment Manager using IBM HTTP Server Admin Process for tasks such as starting, stopping, or automatically pushing the plug-in configuration file.

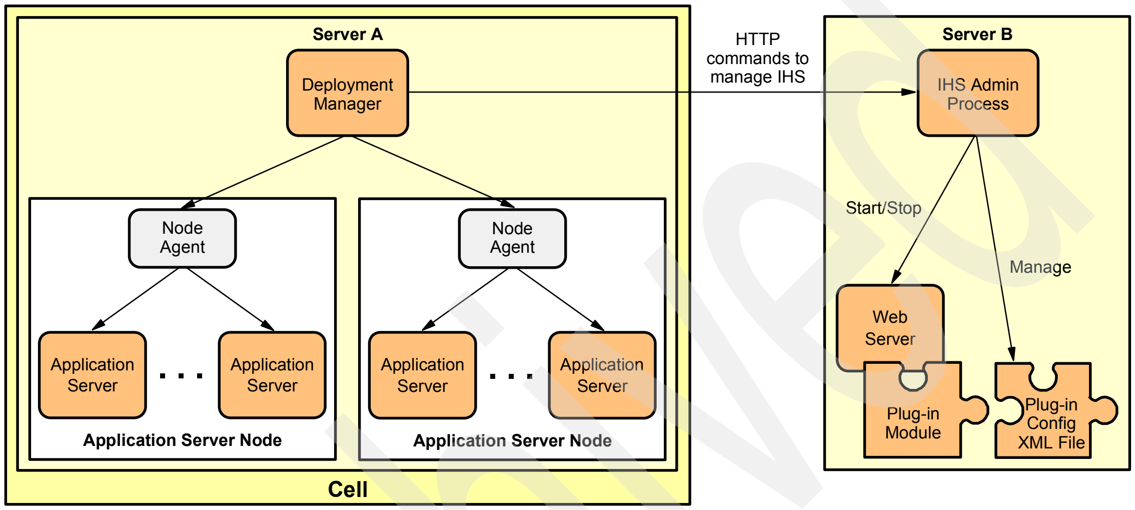


Figure 9-15 IBM HTTP Server Unmanaged Node

9.4.4 Software requirements

Table 9-4 indicates the minimum software components that you need to install to configure the Web server topology in a Network Deployment cell. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-4 Minimum software required

Product / Component Name	Needed
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.5 Mixed node versions in a cell topology

WebSphere Application Server V6 and V5 or later nodes can be part of the same cell. This requires that the Deployment Manager is at V6 level. Those nodes that are at V5 or later level can continue to reside on the same machine where the Deployment Manager at V6 level is installed.

With WebSphere Application Server V6, you can upgrade a portion of the nodes in a cell, while leaving others at the older release level. Thus, for a period of time, you might be managing servers that are running multiple release levels in the same cell.

Figure 9-16 shows a configuration where WebSphere Application Server V6 and V5 or later nodes are part of the same cell.

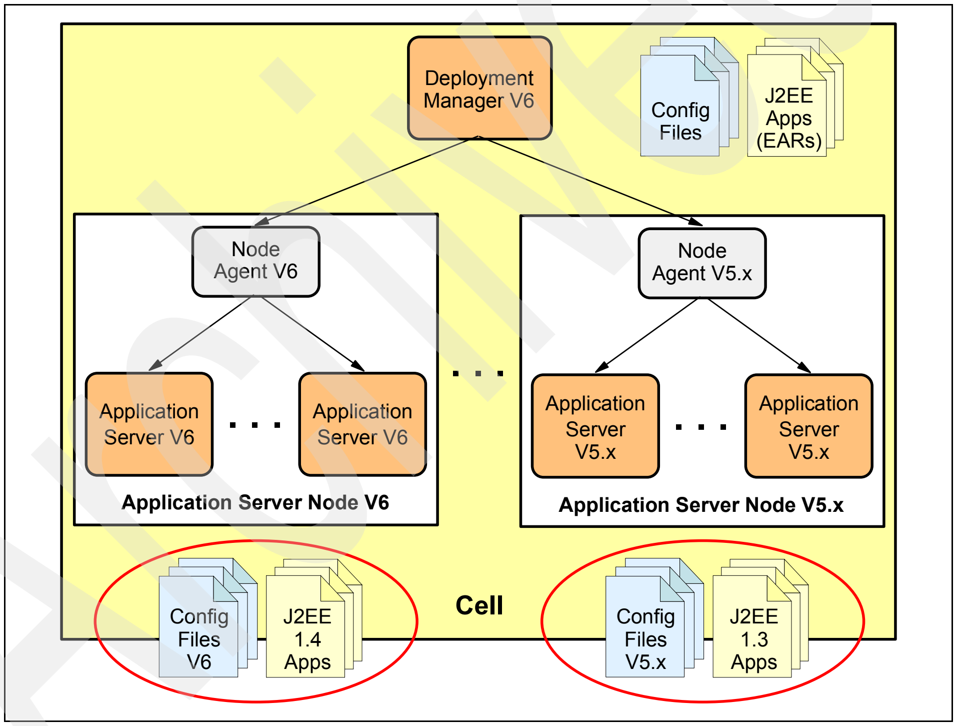


Figure 9-16 Mixed V6 and V5 nodes

9.5.1 Software requirements

Table 9-5 indicates the minimum software components that you need to install to configure the mixed node versions in a cell topology. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-5 Minimum software required

Product / Component Name	Needed
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.6 Clustered topologies

A cluster is a grouping of application servers. Application servers that are inside a cluster are called cluster members. Cluster members run the same set of J2EE applications managed in such way that they behave as a single application server (parallel processing). A cluster provides scalability and failover support. A cell can have zero or more clusters. It can span machine or LPAR boundaries (vertical scalability and horizontal scalability). It can span different operating systems, but cannot span from distributed to z/OS. All cluster members that are part of the cluster must belong to the same cell.

Other than the fact that the Node Agent is not notified of failed servers on other nodes, the impact of a down Deployment Manager in this topology is minimal. If the Deployment Manager is down (failed or stopped), cluster members on the cluster continue to have routed requests. A performance degradation is possible if the Deployment Manager remains down. If the Deployment Manager is down, the Node Agent is not going to be aware of configuration changes, and neither console nor **`wsadmin`** is available in the Deployment Manager node. High availability is improved in WebSphere Application Server V6 through the high availability manager. (See Chapter 13, “Planning for system management” on page 371 for further details about the high availability manager.)

Figure 9-17 shows a cluster that has four cluster members. Those cluster members belong to two different application server nodes (but they still serve the same J2EE application). On each node there is also an application server that is in the same cell but is not part of the cluster (not a cluster member). Therefore, not all the application servers in the node have to be part of the cluster.

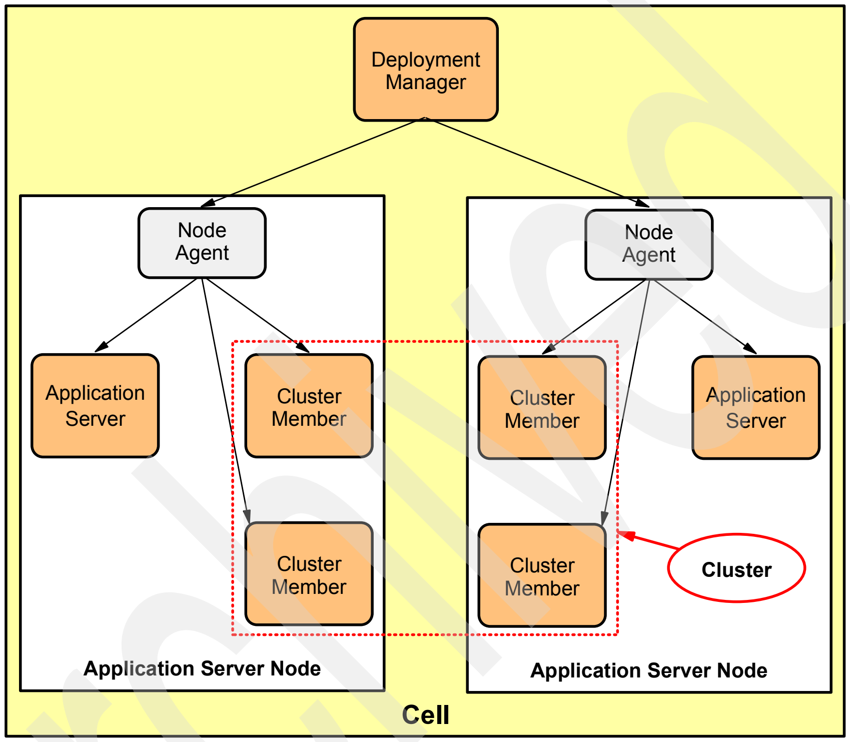


Figure 9-17 Cluster and cluster member

9.6.1 One cluster on one machine

All cluster members can reside on the same machine. This topology is known as *vertical scaling*. For more details about vertical scaling, see 9.8, “Vertical scaling topology” on page 274.

9.6.2 One cluster on multiple machines

Cluster members can also reside sprayed out across different machines. In this topology, each machines has an application server node holding a cluster member. In addition, each application server node has only one Node Agent that is managed by only one Deployment Manager. This topology is known as *horizontal scaling*. For more details about horizontal scaling, see 9.9, “Horizontal scaling topology” on page 276. The combination of vertical and horizontal scaling is possible.

9.6.3 Software requirements

Table 9-6 indicates the minimum software components that you need to install to configure the clustered topologies. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-6 Minimum software required

Product / Component Name	**Needed**
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.7 Mixed node versions in a clustered topology

This topology combines mixed node versions in a cell topology and clustered topologies discussed in the previous sections. This configuration integrates those two concepts into one topology that has mixed application server node versions within a cell and, therefore, mixed cluster members within a cluster. The Deployment Manager has to be at V6 level. Figure 9-18 illustrates a cluster where some nodes are at V6 level and others are at V5 or later level.

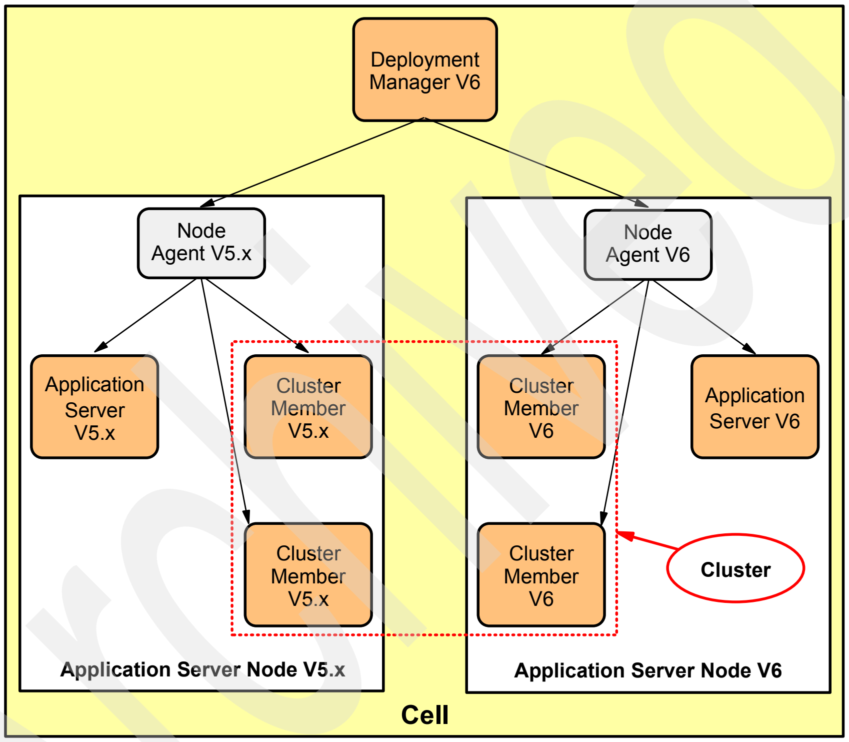


Figure 9-18 Mixed version cluster nodes in a cell

With WebSphere Application Server V6, you can upgrade a portion of the nodes in a cell, while leaving others at a previous release level. Thus, for a period of time, you might be managing servers that are at multiple release levels within the same cell. This situation can occur in a migration process.

In this mixed environment, consider the following restrictions on what you can do with clusters and cluster members:

- You cannot create a cluster member using a server that is running on WebSphere Application Server V5 or later.
- You cannot add a cluster member that is running WebSphere Application Server V5 or later to any cluster.
- You can add a new WebSphere Application Server V6 cluster member to a mixed cluster only if the cluster already contains a WebSphere Application Server V6 member that was upgraded from WebSphere Application Server V5 or later. All new cluster members must be running WebSphere Application Server V6.

9.7.1 Software requirements

Table 9-7 indicates the minimum software components that you need to install to configure the mixed node versions in a clustered topology. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-7 Minimum software required

Product / Component Name	**Needed**
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.8 Vertical scaling topology

Vertical scaling (depicted in Figure 9-19) refers to configuring multiple application servers on a single machine and create a cluster of associated application servers all hosting the same J2EE application(s).

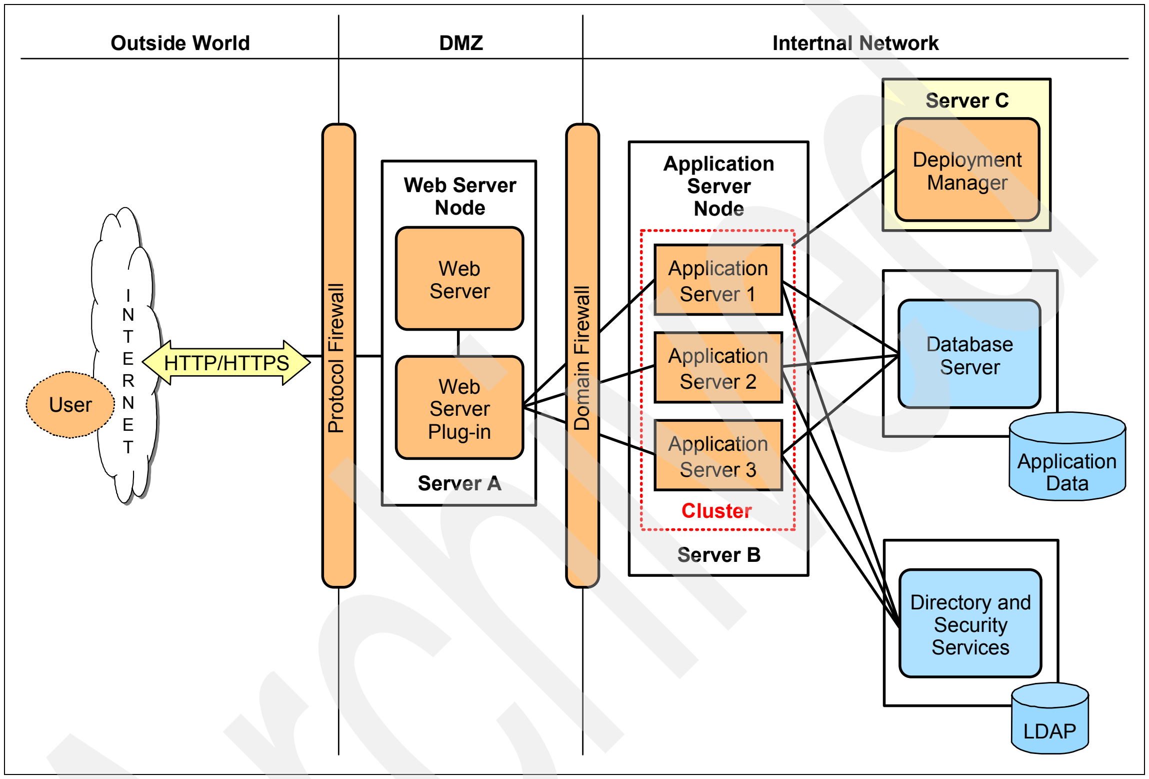


Figure 9-19 Vertical scaling

This vertical scaling example includes a cluster and three cluster members. In this case, the Web server plug-in routes the requests according to the application server's availability. Some basic load balancing is performed at the Web server plug-in level based on a round-robin algorithm.

Vertical scaling can be combined with other topologies to boost performance, throughput and availability.

Advantages

Vertical scaling has the following advantages:

- Efficient use of machine processing power

  With vertical scaling, each application server (server1, server2, and so forth) runs its own Java Virtual Machine (JVM). Each JVM takes a part of the processor and a part of the memory. Because of JVM architecture and application design, sometimes, just one JVM can fully use neither all of the processor nor all of the memory. In these cases, more application servers can be created so that each one starts its own JVM that uses more processor and memory and that improves performance and throughput.

- Load balancing

  Vertical scaling topologies can make use of WebSphere workload management.

- Process failover

  Vertical scaling can provide failover support among application servers of a cluster. If one application server process goes offline, the other one continues processing client requests.

Disadvantages

Single machine vertical scaling topologies have the drawback of introducing the host machine as a single point of failure in the system.

9.8.1 Software requirements

Table 9-8 indicates the minimum software components that you need to install to configure the vertical scaling topology. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-8 Minimum software required

Product / Component Name	needed
WebSphere Application Server - Express or WebSphere Application Server	no
WebSphere Application Server Network Deployment	yes
IBM HTTP Server	yes
Application server plugin	yes
WebSphere Edge Components	no

9.9 Horizontal scaling topology

Horizontal scaling exists when the cluster members are located across multiple machines. This topology lets a single application span over several machines, yet still presents as a single logical image. Figure 9-20 illustrates an horizontal scaling topology with two application server nodes, each one on separated machines (Server B and Server C). There is a fourth server, Server D, where the Deployment Manager is installed to manage the cluster.

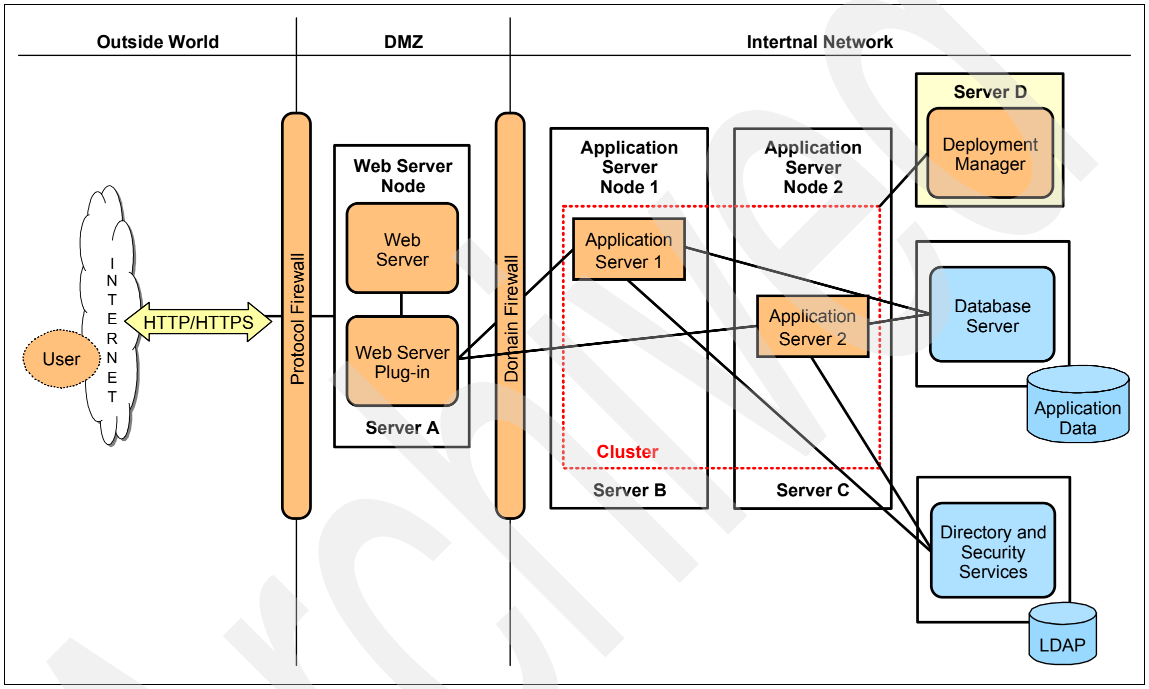


Figure 9-20 Horizontal scaling with cluster

The Web server plug-in distributes requests to the cluster members on each server performing some basic load balancing and offering an initial failover. If the Web server (Server A) goes down, then the embedded WebSphere Application Server Web server of Server B or Server C could be used (limited throughput) meanwhile Server A or the Web server on Server A is repaired.

Another possibility is to install another Web server on either (or both) Server B or Server C, have it configured and off line (stand-by) ready to go to production in the event of any failure on the Web server node.

If any component in the application server node 1 (hardware or software) fails the application server node 2 can serve requests from the Web server node and vice versa.

The Network Dispatcher, part of the Edge Components, can be configured to create a cluster of Web servers and add it to a cluster of application servers. See 9.10, "Horizontal scaling with IP sprayer topology" on page 278 for more information.

Advantages

Horizontal scaling using clusters has the following advantages:

- Improved throughput

  The use of clusters enables the handling of more client requests simultaneously.

- Improved performance

  Hosting cluster members on multiple machines enables each cluster member to make use of the machine's processing resources, avoiding bottlenecks and improving response time.

- Hardware failover

  Hosting cluster members on multiple machines isolates hardware failures and provides failover support. Client requests can be redirected to cluster members on other machines if a machine goes offline.

- Application software failover

  Hosting cluster members on multiple nodes isolates application software failures and provides failover support if an application server stops running. Client requests can be redirected to cluster members on other nodes.

Disadvantages

Horizontal scaling using clusters has the following disadvantage:

- Higher hardware costs
- More complex maintenance
- Application servers must be maintained on multiple machines

9.9.1 Software requirements

Table 9-9 indicates the minimum software components that you need to install to configure the horizontal scaling topology. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-9 Minimum software required

Product / Component Name	Needed
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	No

9.10 Horizontal scaling with IP sprayer topology

Load balancing products can be used to distribute HTTP requests among Web servers that are running on multiple physical machines.

The Network Dispatcher, part of the Edge Components, is an IP sprayer that performs intelligent load balancing among Web servers that are based on server availability and workload as the main selection criteria to distribute the requests.

Figure 9-21 on page 279 illustrates a horizontal scaling configuration that uses an IP sprayer on the Load Balancer node to redistribute requests between Web servers on multiple machines.

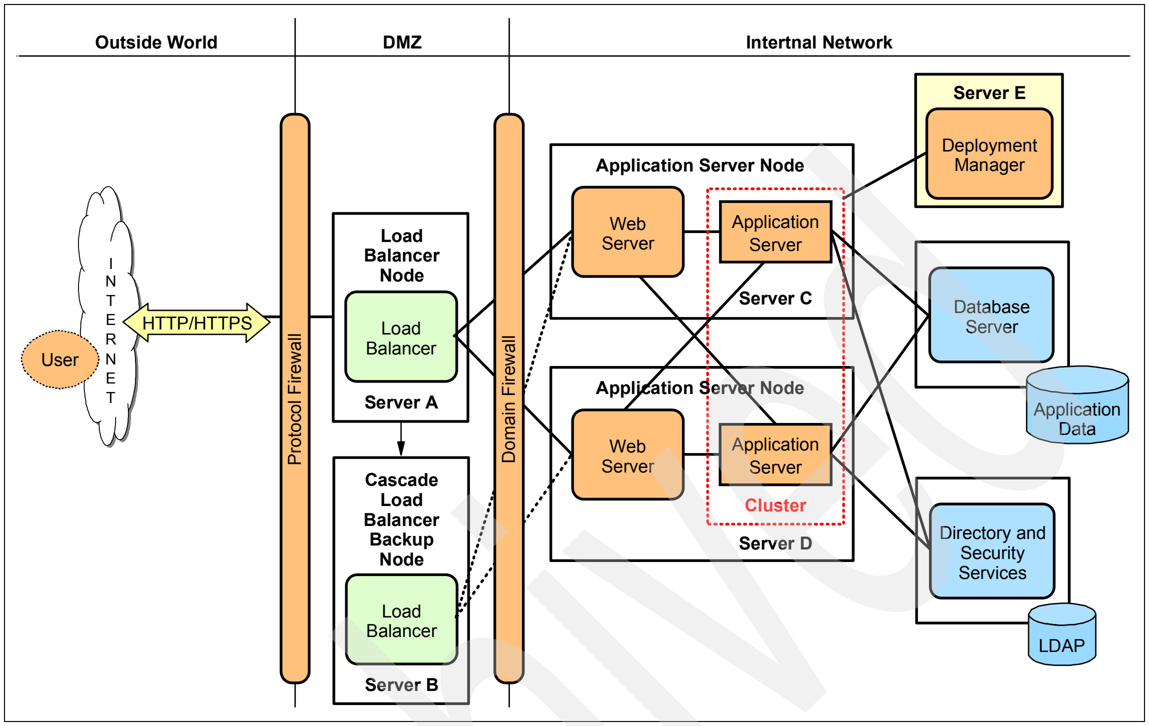


Figure 9-21 Simple IP sprayer horizontally scaled topology

The Load Balancer node sprays Web client requests to the Web servers. The Load Balancer is configured in cascade. The primary Load Balancer communicates to its backup through a heart beat to perform failover, if needed, and eliminates the Load Balancer node as a single point of failure.

Both Web servers perform load balancing and failover between the application servers (cluster members) through the Web server plug-in module and configuration file. If any component on Server C or Server D fails, the other can continue to receive requests.

Advantages

Using an IP sprayer to distribute HTTP requests has the following advantages:

- Improved server performance by distributing incoming TCP/IP requests (in this case, HTTP requests) among a group of Web servers.
- The use of multiple Web servers increases the number of connected users that can be served at the same time.

- Elimination of the Web server as a single point of failure. Used in combination with WebSphere workload management, it eliminates the application servers as a single point of failure.
- Improved throughput and performance by maximizing processor and memory use.

Disadvantages

Using an IP sprayer to distribute HTTP requests means that hardware and software are required for the IP sprayer servers.

9.10.1 Software requirements

Table 9-10 indicates the minimum software components that you need to install to configure the horizontal scaling with IP sprayer topology. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-10 Minimum software required

Product / Component Name	Needed
WebSphere Application Server - Express or WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	Yes

9.11 Topology with redundancy of several components

Having as much redundancy of components as possible eliminates or minimizes the single point of failure. Most components have some kind of redundancy such as a Load Balancer backup node in cascade with the primary Load Balancer node, clustered Web servers or application servers, and so forth. Some other components, such as the Deployment Manager, do not support any automatic or failover node.

Figure 9-22 illustrates a topology with redundancy of several components.

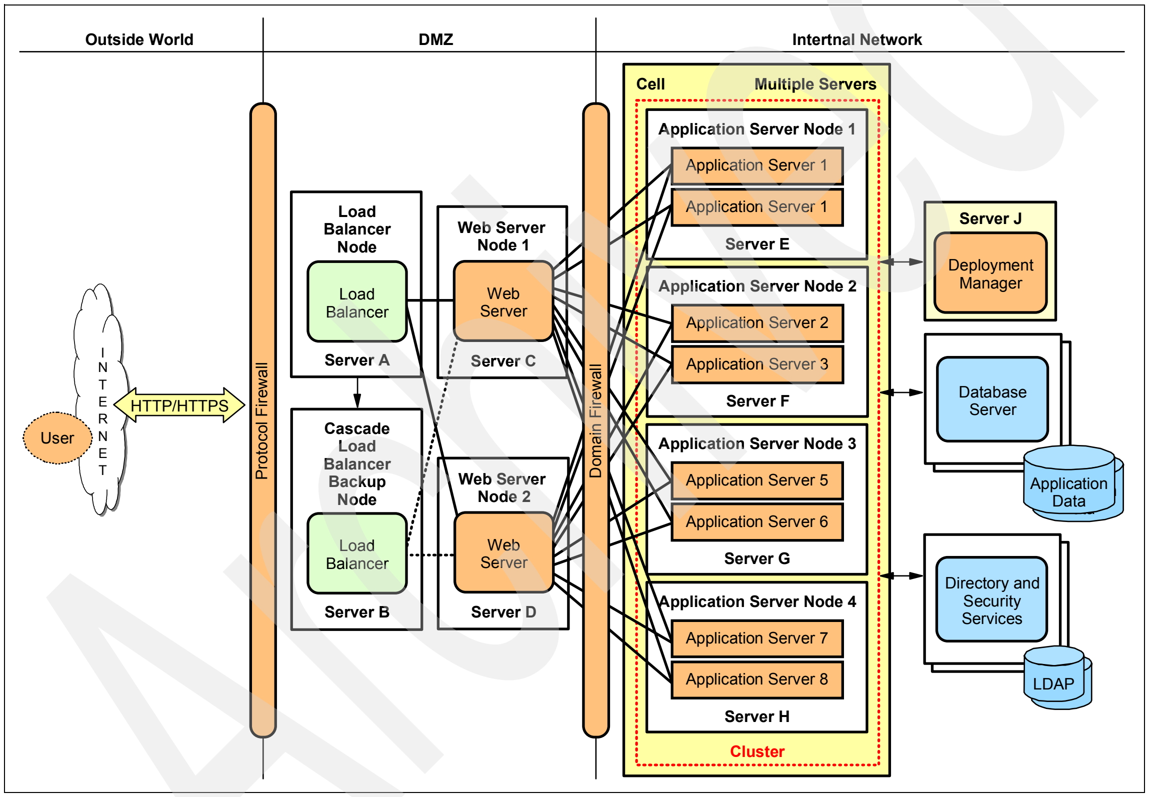


Figure 9-22 Topology with redundancy of several components

The redundant components in this example include:

- Two Load Balancers

  The one on Server A is the primary Load Balancer. It is synchronized, through a heart beat, with a backup Load Balancer in cascade that is in stand by on another machine, Server B.

- Two Web servers

  Both Web servers receive requests from the Load Balancer and share the requests that come from the Internet. Each one is installed on a different machine.

- A cluster

  The cluster implements vertical and horizontal scaling.

- Eight cluster members

  Two on each application server node.

- Four application server nodes

  Each one with two application servers. The nodes on Server E are independent installations. The nodes on Server F are profiles of a single installation.

- Two database servers

  The database servers use a high availability software product. Thus, one copy of the database is being used and the other one is a replica that will replace the first one if it fails.

- Two LDAP servers

  The LDAP servers use a high availability software product. Thus, one copy of the database is being used and the other one is a replica that will replace the first one if it fails.

Advantages

This topology is designed to maximize performance, throughput, and availability. It incorporates the benefits of the other topologies that have been discussed already in this chapter.

- Single point of failure is eliminated from the Load Balancer node, Web server, application server, Database server, and LDAP server due to the redundancy of those components.
- It provides both hardware and software failure isolation. Hardware and software upgrades can be easily handled during off-peak hours.
- Horizontal scaling is done by using both the IP sprayer (for the Web server nodes) and the application server cluster to maximize availability.
- Application performance is improved using the following techniques:
  - Hosting application servers on multiple physical machines to boost the available processing power.
  - Using clusters to scale application servers vertically, which makes more efficient use of the resources of each machine.

- Applications with this topology can make use of workload management techniques. In this example, workload management is performed through:
  - WebSphere Edge Component Network Dispatcher to distribute client HTTP requests to each Web server.
  - WebSphere Application Server Network Deployment workload management feature to distribute work among clustered application servers.

Disadvantages

This combined topology has the disadvantage of costs in hardware, configuration, and administration. You should consider these costs in relation to performance, throughput, and reliability.

9.11.1 Software requirements

Table 9-11 indicates the minimum software components that you need to install to configure redundancy of several components. Considerations about installation and configuration are covered in 10.3, “Selecting components and versions” on page 287.

Table 9-11 Minimum software required

Product / Component Name	Needed
WebSphere Application Server - Express	No
WebSphere Application Server	No
WebSphere Application Server Network Deployment	Yes
IBM HTTP Server	Yes
Application server plugin	Yes
WebSphere Edge Components	Yes

10

Planning for installation

This chapter provides a guide to planning the installation of WebSphere Application Server or any of its components. You should consider topologies, hardware, and software that must be analyzed before starting the installation. This chapter contains the following sections:

- Selecting a topology
- Selecting hardware and operating system
- Selecting components and versions
- TCP/IP ports assignment
- Naming considerations
- Security considerations
- WebSphere Edge Components

10.1 Selecting a topology

Chapter 9, “Topologies” on page 247 covered some common configurations that illustrate single-tier and multi-tier, sometimes complex, environments. Now that you understand these topologies, you need to select the topology that best fits your business requirements.

The topologies that were explained earlier in this book are:

- Single machine topology (stand-alone server)
  - Multiple installs
  - Single install, multiple WebSphere profiles
  - Single machine, single node, one application server
  - Web server separated
- Reverse proxy topology
- Cell topologies
  - Deployment Manager and each node in separate machines
  - Deployment Manager and all nodes in one single machine
  - Multiple nodes on the same machine or LPAR
  - Deployment Manager and one node on the same machine
  - Parts of multiple cells share a physical machine
- Web server topology in a Network Deployment cell
  - Web server Managed Node
  - Web server Unmanaged Node
  - IBM HTTP Server as Unmanaged Node (special case)
- Mixed node versions in a cell topology
- Clustered topologies
  - One cluster on one machine
  - One cluster on multiple machines
- Mixed node versions in a clustered topology
- Vertical scaling topology
- Horizontal scaling topology
- Horizontal scaling with IP sprayer topology
- Topology with redundancy of several components

After identifying the topology that best fits your needs, you map the components from that topology to a specific hardware and operating system.

10.2 Selecting hardware and operating system

After you select the topology, you decide what platforms you will use to map the selected topology to a specific hardware. When you choose the platform or platforms, you can then determine the specific configuration of each server by selecting the processor features, the amount of main memory, and the number of direct-access storage device (DASD) arms and storage space that are required.

Along with selecting the hardware comes the operating system selection. The operating system must be at a supported version with a proper maintenance level installed for WebSphere Application Server to work properly.

You can find an updated list on the hardware and software requirements and supported platforms for WebSphere Application Server V6 at:

`http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html`

10.3 Selecting components and versions

WebSphere Application Server - Express V6, WebSphere Application Server V6, and WebSphere Application Server Network Deployment V6 are incrementally related. As you scale up your e-business demand, you can also scale up your e-business capability by moving from one product to the next.

The Express product is the entry point. With WebSphere Application Server V6, you can host applications on more machines than the Express product. The Network Deployment product includes all the functionality of the other two products. However, with Network Deployment, you can manage multiple machines and processes from a centralized Web application, which is the administrative console of the Deployment Manager.

10.3.1 WebSphere Application Server - Express V6

WebSphere Application Server - Express V6 addresses the basic programming and run-time needs of desktop developers and single-server production scenarios. The run-time environment addresses standards-based programming for Web and component-based programming, as well as Web services.

The administration model for this offering is a single-server environment without clustering, and without centralized administration of multiple server instances.

The basic license for the Express product offering includes installation on two machines. The package also includes a single-use license for the Rational Web Developer tool, which is a fully integrated development environment (IDE).

Installing the Express product offering on the same machine as the Rational Web Developer is not necessary. The IDE contains an exact replica of the WebSphere Application Server V6 as a test environment.

10.3.2 WebSphere Application Server V6

WebSphere Application Server V6 is similar to the Express product but does not have the two-machine license limitation. This offering addresses the basic programming and run-time needs of desktop developers and single-server runtime environments. The runtime environment addresses standards-based programming for Web and component-based programming, as well as Web services.

The administration model is a single-server environment without clustering, and without centralized administration of multiple server instances.

The development environment offering is an unlimited license of the Application Server Toolkit and a trial version of the Rational Application Developer product.

10.3.3 WebSphere Application Server Network Deployment V6

WebSphere Application Server Network Deployment V6 addresses application servers that run in multiple-server production scenarios. The Network Deployment product offering can create:

- Application server profiles

  An application server profile includes default applications and the server1 application server. The application server in the Network Deployment product can run as a managed node or as a stand-alone application server.

  The stand-alone application server is the same as the one in the Express product and in WebSphere Application Server V6 with one important exception: you can add Network Deployment stand-alone application server nodes to a cell under the centralized management of the Deployment Manager.

- Deployment Manager profiles

  The Deployment Manager profile includes the dmgr process. The Deployment Manager provides centralized administration of multiple application server nodes and custom nodes as a single cell. The Deployment Manager provides administration for basic clustering and caching support, including failover support and workload balancing.

- Custom profiles

  A custom profile is an empty node that you must federate. Use the Deployment Manager to customize the node by creating servers and clusters. The node does not include a default application server or default applications.

Table 10-1 shows the application server environments that are created automatically during the product installation.

Table 10-1 Application server environments created during product installation

Product	Default Environment	WebSphere profiles
WebSphere Application Server - Express V6	Stand-alone Application Server	AppSrv01
WebSphere Application Server V6	Stand-alone Application Server	AppSrv01
WebSphere Application Server Network Deployment V6	A default environment is not created during installation of the core product files. The last installation panel lets you launch the profile creation wizard. Use the profile creation wizard to create one of the three available profiles: ► Deployment manager profile ► Custom profile ► Application server profile	None

The installation procedure for WebSphere Application Server - Express V6 and WebSphere Application Server V6 installs the core product files and creates a stand-alone application server in a profile named *default*. The installation procedure for WebSphere Application Server Network Deployment V6 installs the core product files (product binaries) but does not create a profile.

You must start the profile creation wizard and select the type of profile to create. Each use of the profile creation wizard creates one profile.

The product binaries remain unchanged after installation until you install maintenance. All server processes that share the binaries use the updated level of the system files after installing the service.

10.3.4 Planning for Web server installation

Web servers are designed for serving static content. You might want to consider installing a full Web server instead of using the embedded Web server. If you use the embedded Web server, WebSphere Application Server serves the static and the non-static content of your Web application. This could affect the performance of the application.

10.3.5 Planning for Web server plug-in installation

After installing a supported Web server, you should install a WebSphere Application Server plug-in for your Web server. The binary plug-in is unique for each Web server.

The plug-ins installation wizard let you choose between the following supported Web server's plug-ins:

- IBM HTTP Server V6
- Apache Web Server V2
- Lotus Domino Web Server V6 or V6.5
- Sun ONE Web Server 6.0 or Sun Java System Web Server V6.1
- Microsoft Internet Information Services V6
- Microsoft Internet Information Services V5

The plug-ins installation wizard installs the plug-in module, configures the Web server for communicating with the application server, and creates a Web server configuration definition in the application server, if possible.

Installing the plug-in on a remote Web server

Some topologies, such as the Web server on one machine and the application server on another machine, prevent the plug-ins installation wizard from creating the Web server definition in the application server configuration on the remote machine. In such a case, the plug-ins installation wizard creates a script that you can copy to the application server machine and then run the script to create the Web server configuration definition within the application server configuration.

In brief, the steps are:

- The plug-ins installation wizard creates the configureWeb_server_name script in the plugins_installation_root/bin/ directory on the Web server machine.
- The plug-ins installation wizard also creates a temporary plugin-cfg.xml file in the plugins_installation_root/config/Web_server_name directory.

- The script configureWeb_server_name.sh on Linux and UNIX systems or the configureWeb_server_name.bat script on Windows systems must be copied from the Web server's machine to the was_install_root/bin directory on the application server's machine.
- You must run the script to create the Web server definition in the configuration for the application server.
- As soon as the Web server definition is created, the application server creates a plugin-cfg.xml file for the Web server.
- When regeneration occurs, propagate, or copy the actual plugin-cfg.xml file from the application server machine to the Web server machine.

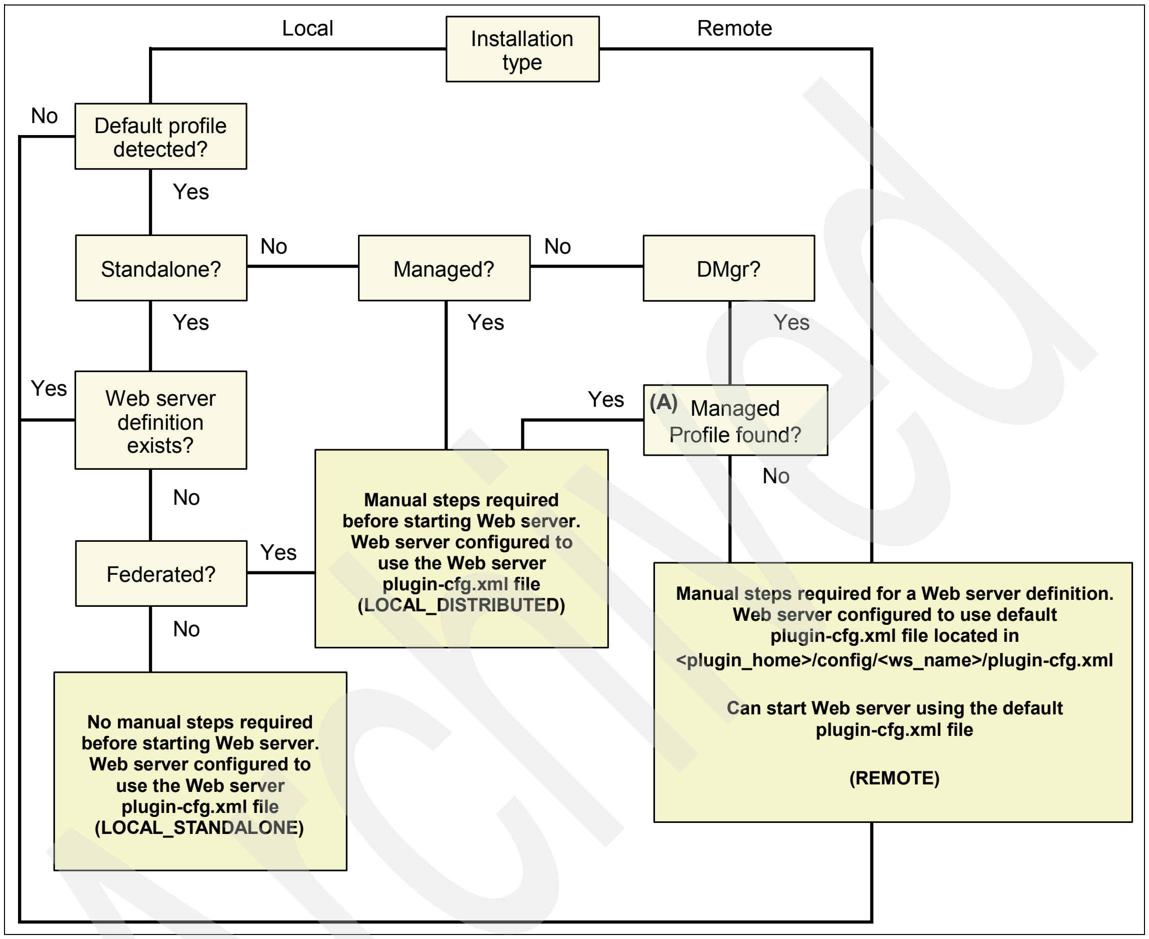
The Web server reads the plugin-cfg.xml file to determine the applications that the application server serves. Whenever the configuration changes, the application server regenerates the file.

WebSphere Application Server products are configured to automatically regenerate the file each time a significant event occurs. Such events include installing applications on the application server and the Web server or creating a new virtual host.

The Web server plug-in configuration service propagates the plugin-cfg.xml file automatically for IBM HTTP Server only. For all other Web servers, propagate the plug-in configuration file by manually copying the plugin-cfg.xml file from the profiles_install_root/config/cells/cell_name/nodes/node_name/servers/web_server_name directory on the application server machine to the plugins_install_root/config/web_server_name directory on Web server machine.

Plug-in installer behavior

Figure 10-1 illustrates the plug-in installer behavior in more detail.



Installation type
Local
Remote
Default profile detected?
No
Yes
Standalone?
No
Managed?
No
DMgr?
Yes
Yes
Yes
Web server definition exists?
Yes
Yes
(A) Managed Profile found?
No
No
Federated?
Yes
No
Manual steps required before starting Web server. Web server configured to use the Web server plugin-cfg.xml file (LOCAL_DISTRIBUTED)
Manual steps required for a Web server definition. Web server configured to use default plugin-cfg.xml file located in <plugin_home>/config/<ws_name>/plugin-cfg.xml
Can start Web server using the default plugin-cfg.xml file
(REMOTE)
No manual steps required before starting Web server. Web server configured to use the Web server plugin-cfg.xml file (LOCAL_STANDALONE)

Figure 10-1 Web server plug-in installer behavior

The plug-in installer maps all possible configurations to three scenarios, LOCAL_STANDALONE, LOCAL_DISTRIBUTED and REMOTE as depicted in Figure 10-1.

- LOCAL_STANDALONE

  What classifies as a LOCAL_STANDALONE plug-in configuration?

  A default unfederated stand-alone profile that has no existing Web server definition is a LOCAL_STANDALONE plug-in configuration.

What tasks does the plug-ins Installation Wizard perform in this case?

- It creates a Web server definition for the default stand-alone profile.
- It configures the Web server to use the plugin-cfg.xml file.

Note: If the stand-alone profile is federated, you will need to re-create the Web server definition.

What's next?

You can start the Web server and WebSphere Application Server without any manual step and hit the snoop servlet to verify that everything is working.

- LOCAL_DISTRIBUTED

  What classifies as a LOCAL_DISTRIBUTED plug-in configuration?

  - A stand-alone profile that has been federated into a Deployment Manager cell.
  - A managed node that is either federated or unfederated in a cell.
  - A managed profile found after a default Deployment Manager cell detected. See (A) in Figure 10-1 on page 292.

  What tasks does the plug-ins installation wizard perform in this case?

  - It does not create a Web server definition for the default distributed profile.
  - It configures the Web server to use the plugin-cfg.xml file in the Web server definition directory that the user needs to create manually. You cannot start the Web server until the manual steps are completed.

  What's next?

  - If the managed node is still not federated, federate the node first. This will avoid the Web server definition being lost after the federation has occurred.
  - Run the manual Web server definition creation script.
  - Start the Web server and WebSphere Application Server and hit the snoop servlet to verify that everything is working.

- REMOTE

  What classifies as a REMOTE plug-in configuration?

  – A remote install type selected by user at install time.
  – A default Deployment Manager profile.
  – No default profiles detected in the WebSphere Application Server directory given by user.
  – A default, unfederated, stand-alone profile with an existing Web server definition.

  What tasks does the plug-ins installation wizard perform in this case?

  – It does not create a Web server definition for the default distributed profile.
  – It configures the Web server to use the plugin-cfg.xml file in <plugin_home>/config/<webserver_name>/plugin-cfg.xml.

  What's next?

  If the user uses the default plugin-cfg.xml file in the <plugin_home> directory, start the Web server and WebSphere Application Server and select the snoop servlet to verify that everything is working.

  To benefit from the Web server definition:

  – Copy the configuration script to the WebSphere Application Server machine.
  – Run the manual Web server definition creation script.
  – Copy the generated Web server definition plugin-cfg.xml file back to the Web server machine into the <plugin_home> directory tree. (For IBM HTTP Server, you can use the propagation feature.)
  – Start the Web server and WebSphere Application Server and select the snoop servlet to verify that everything is working.

10.3.6 Planning for application server clients installation

In a traditional client-server environment, the client requests a service and the server fulfills the request. Multiple clients use a single server. Clients can also access several different servers. This model persists for Java clients except that now these requests use a client run-time environment.

In this model, the client application requires a servlet to communicate with the enterprise bean, and the servlet must reside on the same machine as the WebSphere Application Server.

The application client for WebSphere Application Server V6 consists of the following client applications:

- J2EE application client application (uses services provided by the J2EE Client Container)
- Thin application client application (does not use services provided by the J2EE Client Container)
- Applet application client application
- ActiveX to EJB Bridge application client application

The application client is packaged with the following components:

- IBM Java Runtime Environment (JRE) or an optional full Software Development Kit.
- WebSphere Application Server run time for J2EE application client applications or thin application client applications.
- An ActiveX to EJB Bridge run time for ActiveX to EJB Bridge application client applications (only for Windows).
- IBM plug-in for Java platforms for Applet client applications (Windows only).

Note: The pluggable application client is a kind of thin application client. However, the pluggable application client uses a Sun JRE and Software Development Kit instead of an IBM JRE and Software Development Kit)

The ActiveX application client model, uses the Java Native Interface (JNI) architecture to access the Java virtual machine (JVM) API programmatically. Therefore, the JVM code exists in the same process space as the ActiveX application (Visual Basic, VBScript, or Active Server Pages files) and remains attached to the process until that process terminates.

In the Applet client model, a Java applet embeds in a HyperText Markup Language (HTML) document residing on a remote client machine from the WebSphere Application Server. With this type of client, the user accesses an enterprise bean in the WebSphere Application Server through the Java applet in the HTML document.

The J2EE application client is a Java application program that accesses enterprise beans, Java DataBase Connectivity (JDBC) APIs, and Java Message Service message queues. The J2EE application client program runs on client machines. This program follows the same Java programming model as other Java programs; however, the J2EE application client depends on the Application Client run time to configure its execution environment, and uses the Java Naming and Directory Interface (JNDI) name space to access resources.

The pluggable and thin application clients provide a lightweight Java client programming model. These clients are useful in situations where a Java client application exists but the application needs enhancements to use enterprise beans, or where the client application requires a thinner, more lightweight environment than the one offered by the J2EE application client. The difference between the thin application client and the pluggable application client is that the thin application client includes a Java virtual machine (JVM) API, and the pluggable application client requires the user to provide this code. The pluggable application client uses the Sun Java Development Kit, and the thin application client uses the IBM Developer Kit for the Java platform.

The J2EE application client programming model provides the benefits of the J2EE platform for the Java client application. Use the J2EE application client to develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

For more details about WebSphere Application Server clients, see Chapter 11, “Planning for application development” on page 305.

10.4 TCP/IP ports assignment

The port assignment scheme for a WebSphere site should be developed in close cooperation with the network administrators. From a security point-of-view, it is highly desirable to know each and every port usage beforehand, including the process names and owners using them.

Depending on the chosen WebSphere Application Server configuration and hardware topology, the setup even for a single machine could involve having multiple cells, nodes, and server profiles in a single process space. Each WebSphere process requires exclusive usage of several ports and knowledge of certain other ports for communication with other WebSphere processes.

To simplify the installation and provide transparency to the ports utilization, the following approach is reliable and reduces the complexity of such a scenario considerably:

- Discuss and decide upon with the network administration a fixed range of continuous ports for exclusive use for the WebSphere Application Server installation.
- Draw the overall WebSphere Application Server topology and map your application life cycle stages onto WebSphere profiles.
- List the number of required ports per WebSphere profile and come up with an enumeration scheme, using appropriate increments at the cell, node,

application server level, starting with your first cell. You can use a spreadsheet and develop this spreadsheet as part of your installation documentation. The same spreadsheet also can serve for the server names, process names, user IDs, and so forth.

Table 10-2 shows the default port definitions for WebSphere Application Server V6.

Table 10-2 Default port definitions for WebSphere Application Server V6

Port	Value	File
HTTP transport port (HTTP_TRANSPORT)	9080	serverindex.xml and virtualhosts.xml
HTTPS Transport Port (HTTPS_TRANSPORT)	9443	
HTTP Admin Console Port (HTTP_TRANSPORT_ADMIN)	9060	
HTTPS Admin Console Secure Port (HTTPS_TRANSPORT_ADMIN)	9043	
Internal JMS Server (JMSSERVER_SECURITY_PORT)	5557	serverindex.xml
Service integration port (SIB_ENDPOINT_ADDRESS)	7276	serverindex.xml
Service integration secure port (SIB_ENDPOINT_SECURE_ADDRESS)	7286	
Service integration MQ interoperatibility port (SIB_MQ_ENDPOINT_ADDRESS)	5558	
Service integration MQ interoperatibility secure port (SIB_MQ_ENDPOINT_SECURE_ADDRESS)	5578	
Bootstrap port (BOOTSTRAP_ADDRESS)	2809	serverindex.xml
SOAP connector port (SOAP_CONNECTOR_ADDRESS)	8880	serverindex.xml
Data Replication Service port (DRS_CLIENT_ADDRESS)	7873	serverindex.xml
SAS SSL ServerAuth port (SAS_SSL_SERVERAUTH_LISTENER_ADDRESS)	9401	serverindex.xml
CSIV2 SSL ServerAuth listener port (CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS)	9403	serverindex.xml
CSIV2 SSL MultiAuth listener port (CSIV2_SSL_MULTIAUTH_LISTENER_ADDRESS)	9402	
High availability manager communication port	9353	serverindex.xml

Port	Value	File
ORB listener port (ORB_LISTENER_ADDRESS)	9100	serverindex.xml
IBM HTTP Server Port	80	virtualhosts.xml, plugin-cfg.xml, and IHSinstall_root/conf/ httpd.conf
IBM HTTP Server Admin Port	8008	IHSinstall_root/conf/ admin.conf
NODE_MULTICAST_IPV6_DISCOVERY_ADDRESS	5001	serverindex.xml

10.5 Naming considerations

The purpose of developing systematic naming concepts and rules for a WebSphere site is two-fold.

- To provide guidance during setup and configuration

  Clarifying functional and non-functional requirements beforehand and documenting key decisions results in a more stable and ready-for-growth WebSphere Application Server site.

- In case of issues, to quickly narrow down the source of the issue

  You can reduce the complexity of a site installation by answering questions regarding the type of information, whether it is a valid configuration item, and where it belongs or comes from.

Spending some extra time on the application-related naming concepts quickly pays off in practice, because it further reduces the time spent on analyzing the source of issues during standard operations of future J2EE applications. From an automation point-of-view, it is a very good idea to ensure that the concepts developed here are consistent and self-contained and can be easily implemented by easy-to-maintain algorithms. This avoids the cumbersome overhead of manual maintenance of mapping tables between the different naming concepts used in different scripts and places.

10.5.1 Naming concepts

Developing, establishing, and maintaining good naming for all of the setup and configuration information of a WebSphere Application Server site is an important part of the installation planning. All parts of the installation planning, particularly the naming concepts, should be discussed with the subject matter experts involved, and the decisions and results documented in writing. Failure to do so can affect operational processes for the future site.

Some considerations to be taken into account when developing the key concepts for the site during the installation planning are:

- Naming for infrastructure elements

  Make sure you can map applications, resources, security info, and configuration parameters with a uniform naming scheme.

- Naming hosts and domains

  Be sure to comply with the Internet standards for host and domain naming. These standards specify that hostnames should not contain underscores (RFC 1123 & RFC 932). Domain names must have at least one dot or period. This is an internet standard (RFC 2109). Do not use the server name (such as machinename.ibm.com® or hostname.ibm.com) in the domain name.

- Naming profiles

  The profile name can be any unique name. However, do not use any of the following characters when naming your profile:

  - Spaces
  - Illegal special characters that are not allowed within the name of a directory on your operating system (namely * & ? ‘ “, and so forth)
  - Slashes (/) or (\)

- Reserved names

  Avoid using reserved folder names as field values. The use of reserved folder names can cause unpredictable results. The following words are reserved:

  - Cells
  - Nodes
  - Servers
  - Clusters
  - Applications
  - Deployments

- Object names

  When you create a new object using the administrative console or a **`wsadmin`** command, you often must specify a string for a name attribute. Most characters are allowed in the name string (numbers, letters). However, the name string cannot contain special characters or signs. The dot is not valid as first character. The name string also cannot contain leading and trailing spaces.

10.6 Security considerations

Security has an important role when planning the installation. The following is a list of security considerations that you should take into account during the installation planning phase:

- If you will use digital certificates, they must be requested with enough time.
- If default certificates or dummy key ring files are provided with any of the products you plan to install, replace them by your own certificates.
- User profiles with enough authority for the installation purpose should be requested (root on a Linux or UNIX operating system and a member of the administrator group on a Windows operating system).
- If there is a policy on passwords expiration, it should be well know to avoid disruption on the service. (Password expiration of root, QSECOFR, Administrator or the password of the user to access some database).
- Usually a firewall is part of the topology. After determining what ports need to be open a request to the firewall(s) administrator to open them must be performed with enough time.
- Physical access to the data center where the machines are going to be installed should be planned to prevent delays to the personnel involved in the installation and configuration tasks.
- After the basic topology has been created, consider enabling Global Security and secure WebSphere Application Server console to avoid loosing control of the configuration process. WebSphere Application Server console has the following roles. Plan which person covers each one.
  - Administrator

    The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.
  - Configurator

    The configurator role has monitor permissions and can change the WebSphere Application Server configuration.
  - Operator

    The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

  - Monitor

    The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

For more details about securing your installation, refer to Chapter 17, “Planning for security” on page 487.

10.7 WebSphere Edge Components

This book does not cover WebSphere Edge Components. However, because it mentions them, this section provides a brief explanation of their features.

You can use Edge Components in conjunction with WebSphere Application Server to control client access to Web servers and to enable business enterprises to provide better service to users who access Web-based content over the Internet or a corporate intranet. Using Edge Components can reduce Web server congestion, increase content availability, and improve Web server performance.

As the name indicates, Edge Components usually run on machines that are close (in a network configuration sense) to the boundary between an enterprise’s intranet and the Internet. Edge Components are:

- Caching Proxy
- Load Balancer

10.7.1 Caching Proxy

Caching Proxy reduces bandwidth use and improves a Web site’s speed and reliability by providing a point-of-presence node for one or more back-end content servers. Caching Proxy can cache and serve static content and content dynamically generated by WebSphere Application Server. The proxy server intercepts data requests from a client, retrieves the requested information from content-hosting machines, and delivers that content back to the client.

Most commonly, the requests are for documents stored on Web server machines (also called origin servers or content hosts) and delivered using the Hypertext Transfer Protocol (HTTP). However, you can configure the proxy server to handle other protocols, such as File Transfer Protocol (FTP) and Gopher. The proxy server stores cachable content in a local cache before delivering it to the requester. Examples of cachable content include static Web pages and JavaServer Pages files that contain dynamically generated, but infrequently changing, information.

Caching enables the proxy server to satisfy subsequent requests for the same content by delivering it directly from the local cache, which is much quicker than retrieving it again from the content host.

Plug-ins for Caching Proxy add functionality to the proxy server.

- The ICP plug-in enables the proxy server to query caches that are Internet Caching Protocol compliant in search of HTML pages and other cachable resources.
- The Tivoli Access Manager (formerly Policy Director) plug-in enables the proxy server to use Tivoli Access Manager's integrated authorization or authentication services.
- The PAC-LDAP Authentication Module enables the proxy server to access an LDAP server when performing authorization or authentication routines.
- The WebSphere Transcoding Publisher plug-in allows the proxy server to cache multiple transcoded versions of content for mobile devices when used in conjunction with WebSphere Transcoding Publisher.

You can further extend the functions of Caching Proxy by writing custom plug-in modules to an API. The API is flexible, easy to use, and platform independent. The proxy performs a sequence of steps for each client request it processes. A plug-in application modifies or replaces a step within the request-processing workflow, such as client authentication or request filtering. The powerful Transmogrify interface, for example, provides access to HTTP data and enables substitution or transformation of URLs and Web content. Plug-ins can modify or replace designated processing steps, and you can invoke more than one plug-in for a particular step.

10.7.2 Load Balancer

Load Balancer creates edge-of-network systems that direct network traffic flow, reducing congestion and balancing the load on various other services and systems. Load Balancer provides site selection, workload management, session affinity and transparent failover.

Load Balancer is installed between the Internet and the enterprise's back-end servers, which can be content hosts or Caching Proxy machines. Load Balancer acts as the enterprise's single point-of-presence node on the Internet, even if the enterprise uses multiple back-end servers because of high demand or a large amount of content. You can also guarantee high availability by installing a backup Load Balancer to take over if the primary one fails temporarily.

Load Balancer intercepts data requests from clients and forwards each request to the server that is currently best able to fill the request. In other words, it balances the load of incoming requests among a defined set of machines that service the same type of requests. Load Balancer can distribute requests to many types of servers, including WebSphere Application Server's and Caching Proxy machines.

Load balancing can be customized for a particular application or platform by using custom advisors. Special purpose advisors are available to obtain information for load balancing WebSphere Application Server.

If the Content Based Routing component is installed together with the Caching Proxy, HTTP and HTTPS requests can even be distributed based on URLs or other administrator-determined characteristics, eliminating the need to store identical content on all back-end servers. The Dispatcher component can also provide the same function for HTTP requests.

Load balancing improves your Web site's availability and scalability by transparently clustering content servers, including HTTP servers, application servers, and proxy servers, which are surrogate content servers. Availability is achieved through parallelism, load balancing, and failover support. When a server is down, business is not interrupted. An infrastructure's scalability is greatly improved because back-end processing power can be added transparently.

Load Balancer includes the following components:

- Dispatcher

  For all Internet services, such as HTTP, FTP, HTTPS, and Telnet, the Dispatcher component performs load balancing for servers within a local area network (LAN) or wide area network (WAN). For HTTP services, Dispatcher can perform load balancing of servers based on the URL content of the client request. The Dispatcher component enables stable, efficient management of a large, scalable network of servers. With Dispatcher, you can link many individual servers into what appears to be a single virtual server. Your site thus appears as a single IP address to the world.

- Content Based Routing

  For HTTP and HTTPS services, the Content Based Routing component performs load balancing for servers based on the content of the client request. The Content Based Routing component works in conjunction with the application server Caching Proxy component.

- Site Selector

  The Site Selector component enhances a load-balancing system by allowing it to act as the point-of-presence node for a network and load balance incoming requests by mapping DNS names to IP addresses. In conjunction with Metric Server, Site Selector can monitor the level of activity on a server, detect when a server is the least heavily loaded, and detect a failed server.

- Cisco CSS Controller

  The Cisco CSS Controller component generates server-weighting metrics that are sent to a Cisco CSS switch for server selection, load optimization, and fault tolerance.

- Nortel Alteon Controller

  The Nortel Alteon Controller component generates server-weighting metrics that are sent to a Nortel Alteon switch for server selection, load optimization, and fault tolerance.

- Metric Server

  The Metric Server component runs as a daemon on a load-balanced server and provides information about system loads to Load Balancer components.

10.7.3 Installing and configuring Edge Components

For information about installing and configuring Edge Components, visit IBM WebSphere Application Server Edge Components Information Center:

http://www.ibm.com/software/webservers/appserv/ecinfocenter.html

The most current information about hardware and software requirements for Edge Components is available in following Web page:

http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html

11

Planning for application development

This chapter provides information about what to consider when planning a development environment and processes with WebSphere Application Server V6 Software. This chapter contains the following sections:

- Versioning
- Versioning strategies and best practices
- End-to-end life cycle
- General development best practices
- Java development best practices
- Enterprise Java development best practices

11.1 Versioning

In development, it is important to manage generations of code. Application builds and the source code used to create builds has to be organized and tracked carefully. If it is not, then confusion can arise as to what level of code was used to create which build of the application. Generally, some form of the VRMF (Version, Release, Modification, Fix) schema is used to organise code and builds in a dotted number system such as Java SE 1.4.2.5. In this way code management systems can be certain of identifying, creating and recreating application builds accurately from the correct source code.

This section provides information to help you in planning for a successful versioning and version control. First, it looks at the Source Code Management (SCM) systems available for version control in the WebSphere Application Server V6 environment. It considers which software to use and configuration issues that must be thought of when planning a development environment. Finally, it looks at best practice in version control within this environment.

11.1.1 Source Code Management systems

This section looks at SCM systems — what they are, why they are needed, which are available, and which should be used.

Note: The acronym SCM has a number of related definitions. In addition to Source Code Management it is also defined as Software Configuration Management. The latter is a super set of the former. It does more than just manage code. This chapter assumes the definition as first stated.

What is an SCM system

Developers produce code. They can use a simple editor and compiler or they can use an IDE such as Rational Application Developer for producing code. Code in Rational Application Developer is organized under projects such as EJB or Web projects and stored in a work space on the file system. Code produced by a developer not using an IDE is also typically stored to the file system. This code base soon grows and needs to be managed centrally. A centralized repository of code is required for:

- Development team collaboration (work on common code)
- Code versioning (managing which versions are in which releases)
- Wider team collaboration (access for project managers, testers)

SCM systems are used for these purposes. The SCM systems supported by Rational Application Developer are:

- Rational ClearCase
- Concurrent Versions System (CVS)

Rational ClearCase

Rational ClearCase organizes its code repositories as Versioned Object Bases or VOBs. VOBs contain versioned file and directory elements. Users of Rational ClearCase are organized according to their role. Each user has their own View of the data that is in the VOB on which they are working. Rational ClearCase tracks VOBs and Views and coordinates the checking in and checking out of VOB data to and from Views.

As the role based model suggests, Rational ClearCase is not just an SCM system but also a Software Asset Management (SAM) system. This means that it not only manages code but other assets. These further assets might be produced by the other Rational products with which Rational ClearCase integrates.

The Rational products with which ClearCase integrates are Rational Enterprise Suite Tools, the Rational Unified Process and, of course, Rational IDEs. Artifacts such as use cases generated by Rational RequisitePro can be stored in Rational ClearCase. These can then be fed into a Rational Rose design model and used to design Java components and generate Unified Modeling Language (UML) diagrams and documentation.

ClearCase can also be used to implement the Unified Change Management (UCM) process, this change management process can be enhanced by using Rational ClearCase in conjunction with Rational Rational ClearQuest which is change and defect tracking software.

The software is scalable. Rational ClearCase LT is a cut down version of Rational ClearCase for small-to medium-sized teams that can be upgraded seamlessly to Rational ClearCase as a user’s needs change. Additionally a ClearCase MultiSite® add on can be obtained to support use of the software in geographically dispersed development teams.

In short, while ClearCase is an Source Code Management system it is also an integral part of the Rational toolset and the Rational Unified Process.

For more information about Rational software, see:

http://www.ibm.com/software/rational

Concurrent Versions System

Concurrent Versions System (CVS) has the following features:

- ► Free to use under the GNU license
- ► Open source
- ► Widely used in the development community
- ► Other SCM repositories can be converted to CVS
- ► Many free client applications are available, for example WinCVS
- ► Can store text and binary files
- ► Handles versioning and branching
- ► Is a centralized repository

For more information about Concurrent Versions System, see :

http://www.cvshome.org

11.1.2 Which SCM should you use

The obvious question arises: which SCM should the team use? There is no simple answer to this question, because the answer depends on a number of factors such as:

- ► Current software and processes
- ► Team size
- ► Complexity of requirements
- ► Cost
- ► Are the above factors likely to change

Current software and processes

To some extent, the choice depends on what the legacy situation is (if any) and what the SCM and development process requirements are now and in the future. If a team uses CVS and an existing, successful, development process, then ClearCase might not be necessary, especially if the size and complexity of requirements is not likely to grow in the future. If this is not the case, then Rational ClearCase LT or Rational ClearCase would be a good choice so that the full integration of Rational and WebSphere products can be exploited now and in the future.

Team size

Rational ClearCase LT gives a sound starting place for smaller teams. Rational ClearCase LT can be upgraded to Rational ClearCase later if necessary. On very large development projects, Rational ClearCase and Rational ClearQuest have a MultiSite option that allows for easier development by geographically dispersed development teams.

Complexity of requirements

The Rational Unified Process (RUP®) provides a holistic approach to the end-to-end development life cycle. The use of the UCM process, which is part of the RUP, can shield the user from complex tagging and branching of code. While CVS would not shield the user from this.

Cost

CVS is a possibly a cheaper option as it is free and has a large user base which means cheaper skills. In terms of hardware it is likely that hardware costs for hosting CVS itself are cheaper because of its smaller footprint. However, these might be false economies. The limitations of CVS can cause a team to migrate to Rational ClearCase later. CVS does have some short comings. Another GNU project called Subversion is attempting to address these. However Subversion is relatively new and not supported by Rational Application Developer. For more information about Subversion refer to:

`http://subversion.tigris.org`

Migration

Migration from CVS to Rational ClearCase is easier than a migration in the other direction because:

- Rational ClearCase has import tools for importing code from other SCMs such as CVS.
- CVS does not have tools for importing from Rational ClearCase though it does have tools for converting to CVS from other SCM systems such as SCCS and CMS.
- Rational ClearCase is more than a repository for code. It is also a repository for artifacts for the Rational Unified Process and the Rational Enterprise Suite of tools. When Rational Unified Processes and their related tools are used extensively, it makes less sense and becomes far more problematic to migrate to CVS.
- CVS cannot store some Rational artifacts such as Rational Rose models (even in binary format).

Migration in either direction between Rational ClearCase and CVS will have an overhead in terms of training team members and in terms of migrating current development processes.

Summary

In summary, the smaller the development team and the less complex the requirements the more likely that CVS or Rational ClearCase LT are good choices. As team size and complexity grows, Rational ClearCase and then Rational ClearCase MultiSite become more attractive. Legacy processes and software as well as budget for new software, hardware and training are likely to inform the decision further. In matters of cost there might be false economies.

11.1.3 Hardware and network considerations

Whichever SCM system is chosen there are two key best practices:

- For the server use at least one dedicated machine. For security and performance, the system should not host other unrelated processes.
- Use an automated, daily back up strategy to copy repository directories to a separate site machine for disaster contingency.

CVS specific considerations

The following are specific resource needs of a CVS solution:

- Network considerations
  - Clients connect to the CVS repository server on port 2401.
  - This port is the only access to the server needed by the client machines.
  - Secure connections can be used which require additional ports.
  - Back up systems usually need additional connectivity to CVS server.
- Hardware Considerations
  - Hardware for CVS can be relatively modest.
  - Minimum free RAM required by the server is the number of concurrent client users multiplied by 2 MB or the size of the largest source directory (if larger than 2 MB).
  - Minimum free hard disk requirement is three times the source code size expected.
- Operating System Considerations
  - The CVS server runs on all major operating systems: Windows, UNIX, VMS.
  - There are many client applications available for CVS for various platforms. Rational Application Developer has a built in CVS client.
  - The server file system for the repository must have read/write access to it even for reading only by CVS clients as file locking requires write privileges on the server.

Rational ClearCase Specific Considerations

Rational ClearCase LT, Rational ClearCase, and Rational ClearCase MultiSite are the three scaled versions in the Rational ClearCase family. The following are considerations that are specific to this family.

- Network
  - A **release area** on a release host is required. Install images are placed on a host in the network.
  - All Rational ClearCase servers and clients are installed from this release host on to other client and server machines.
  - The site administrator runs a site preparation program against the release area to set up default values for all clients and servers.
  - The server software is installed on one or more server hosts that host VOBs or other server processes such as the registry process.
  - It is possible for a single machine to be a release host, client and server host.
  - Communication between machines containing VOBs and Views is highly network intensive. Extensive use of Remote Procedure calls between these nodes is one of the contributing factors. The topology should ensure that there is a high performance network between all clients and the server hosts. Especially those hosting Views and VOBs.
  - If teams are likely to be geographically dispersed, poor network performance issues can be solved by replicating VOBs using Rational ClearCase MultiSite.
- Hardware
  - Release hosts require 400 MB of free hard disk space.
  - Client host machines require a minimum of 32 MB of free memory and 300 MB hard disk space.
  - Server host machines require a minimum of 64 MB of free memory and 2 GB of disk space.
  - If developers are using Rational Application Developer as the client then that software requires significantly more resource: at least 768 MB of RAM and 3.5 GB of hard disk space for a comprehensive install.

Note: Where possible, it is best practice to stripe logical volumes or partitions containing VOBs across hard-disks using RAID technology in order to increase seek time performance.

- Operation system considerations

  Table 11-1 lists all supported operating systems and platforms.

Table 11-1 Supported platforms and operating systems for Rational ClearCase

Hardware	**Operating System**
Solaris Sparc	Solaris 2.6,7,8,9
HP 9000 series 700 and 800	HP-UX 11.0,11.0 ACE and 11.11
HP IPF	HPUX 11.22
SGI MIPS	IRIX 6.5.12 to 6.5.19 (64 bit only)
IBM pSeries®, RS/6000®	AIX 4.3.3, 5.1, 5.2
IBM PC compatibles	Red Hat Linux 7.2,7.3, 8.0,9 Red Hat Linux Enterprise Linux 2.1, 3 SuSE Linux Standard Server 8 SuSE Linux Enterprise Server 8 Windows NT® 4.0 SP6a and SRP Windows 2000 SP2, SP3 Windwos XP Professional SP1 Windows Server 2003
IBM S/390® and zSeries	Suse Linux Enterprise Server 7,8

- Rational ClearCase LT specific considerations

  Rational ClearCase LT supports a single server host machine. All VOBs are on a single machine.

- Rational ClearCase specific considerations

  Rational ClearCase supports the additional functionality over Rational ClearCase LT:

  – Allows a client to access multiple servers.

  – The multiple server processes can be run on a single machine but it is more usual to run some of the additional servers (for example the VOB servers) on separate machines for better performance.

  The additional server processes are:

  – License server: Allocates licenses to clients on request.
  – Registry server: Keeps a track of all the VOBs and Views in the network.
  – VOB Server machine or machines: Contains one or more VOBs.
  – Rational Web Platform host: Serves the Web interface.

Additional things to consider:

  - When setting up a Rational ClearCase environment the License Server and Registry Server are the server installations that should be set up first from the release host server.
  - The licence Server and Registry Server are easier to maintain and administer if they run on the same machine but this does not have to be the case.
  - The Rational Web Platform host obviously should not have any other Web server software running or there will be port conflicts.

- Considerations specific to Rational ClearCase MultiSite are:
  - Rational ClearCase MultiSite can only run on Rational ClearCase.
  - The key feature of this additional functionality is that it allows the replication and synchronisation of VOBs.
  - When Rational ClearCase MultiSite is installed and used then a VOB can be replicated at multiple sites.
  - VOB replicas allow independent development at multiple sites. Issues of conflicts are controlled by making one replica the master of a particular branch of code.
  - MultiSite only needs to be installed on server hosts that will host VOB replicas.
  - VOB replicas take up the hard disk space that the original VOB takes up plus space to store the packets of data that are used to send updates and receive updates from and to other replicas. This packet size is variable.

Tip: Synchronizing more frequently reduces the packet size. If disk space becomes a problem then a packet size maximum limit is configurable.

Note: Licenses for Rational ClearCase MultiSite have to be available for each developer's client machine that will use replicas. This is often less than the total number of developers using Rational ClearCase as not all developers might need to access replicated VOBs. Licenses for Rational ClearCase also need to be available for each developer using Rational ClearCase whether they are using Rational ClearCase MultiSite or not.

11.2 Versioning strategies and best practices

This section looks at strategies for code management using the software available for SCM.

11.2.1 Rational ClearCase

Rational ClearCase can be used in two fundamental ways:

- The Unified Change Management approach
- The base Rational ClearCase approach

The Unified Change Management approach

Rational ClearCase can be used in conjunction with the Unified Change Management (UCM) process. Rational ClearCase has tools, wizards, and interfaces that support this approach. This approach gives the project manager and the developers an off-the-peg process for code management and versioning.

Terminology

To avoid any misunderstanding because this terminology has many other applications, the following is a list with the terms and definitions this section uses while covering the UCM approach.

- A *project* is an object that contains configuration information needed to manage and track work on a product.
- An *activity* is an object that records the set of files that must be created or modified to complete a development task.
- A *baseline* identifies one version of each element in a component that is the integrated or merged work of the team.
- A *stream* is an object that maintains a list of activities and baselines and determines which versions of elements appear in a view.
- A *work area* consists of a view and a Stream. The view shows a single version of elements and the Stream determines what appears in the view.
- An *integration stream* is the primary, shared stream in a project to which and from which code is added. A project has only one integration stream.
- An *integration view* is a view of the integration stream for each developer.
- A *shared work area* is a work area that more than one developer uses. The primary shared work area for a project contains the integration stream and integration views. It is possible for there to be other, secondary shared work areas where more than one developer works on code that will be integrated in to the integration stream at some point.

- A *development stream* and a *development view* are streams and views specific to a single developer's work area. This constitutes a *private work area*. Private work areas allow for developers to work on code before integrating their work back with work in the integration stream.
- Checking the private work area code into the shared work area is know as *delivering*.
- *Policies* can be set up by the project manager to enforce desirable behavior by team members. *Triggers* can be set up to automate the running of tasks when a particular Rational ClearCase operation is performed.

Scenario for planning and setting up a project

When planning a project for the first time in Rational ClearCase using the UCM process, the project manager would typically:

- Import code files from an existing SCM into Rational ClearCase.
- Create a new project with a Project VOB and a Component VOB.
- Create baselines in the project to indicate starting points.
- Create a shared work area with an integration stream and integration views.
- Create private work areas with development streams and views.
- Update and promote baselines in the shared work area to indicate milestones in the project.
- Set up policies and triggers to enforce good practice and automate tasks.

Developers will work on the code assigned to them in their development streams. They use their development view to see their work in this area. When they finish their work they will deliver their work to the integration stream. They use the integration view to see the integration stream and how their updates have affected the integration stream. Delivery of the code to the integration stream involves merging the code in the development stream with the original base lined code in the integration stream.

When a milestone integration of code has been reached such as a stable version of all elements which are the result of merging one or more development streams with the integration stream code, the Project Manager will create a new base line and promotes it as the recommended baseline. Developers can then update their development work areas with this latest recommended base line.

Tip: The key to ensuring that conflicts and complicated merges are avoided is to firstly allocate development streams with only those items that the developer needs to complete their activity and secondly to create regular baselines and enforce that developers update to the recommended baseline regularly. A good policy is to ensure that developers update their workspace with the latest recommended baseline before they deliver their work to the integration stream. Policies and triggers are tools that can be set up to enforce and automate such practices.

Note: Rational ClearQuest is a change-request management application. It complements Rational ClearCase and the Unified Change Management approach. When using the UCM approach it is then very useful to use Rational ClearQuest as well. This software allows project managers and developers to find assignments and create reports and charts on activities and progress.

The base Rational ClearCase approach

The base Rational ClearCase approach is used when a team wants to implement its own bespoke system of tracking and managing software resources. This brings the project manager and developers much closer to the underlying workings of Rational ClearCase. The UCM approach has a lot of high level concepts which shield the users from these underlying concepts. The underlying concepts are similar in nature to concepts in any Source Code Management system. CVS for example has similar concepts too (although the name for such concepts often differs).

Terminology

Some terminology needs to be defined here also:

- Elements such as files and directories have versions. In the simplest form there is one main *branch* or sequence of these versions: 0,1,2,3,4 etc. However, it might be necessary to work on two separate sequences of code. For example, a file at version 4 can be released to a production environment. Subsequent work on the file for the next release produces versions 5 and 6. A bug that requires an urgent fix is found in production. Working on version 5 of 6 to fix the bug will not work as there are additional features in 5 and 6 that the team does not want to release into production at this moment. Indeed working on the fix on versions 5 and 6 might disrupt work for the next release. Therefore, a new sub branch is created at version 4 of the file. This new branch represents a separate line of development. Sub branches might well have sub branches themselves. In this way an infinite number of development lines can be created.

- Branches can be *merged* back into parent branches or they can continue for indefinitely as a separate branch.
- Element versions can have *version labels* applied to them. The version of the element does not necessarily correspond to the version number in the version label. The version label is usually applied to a number of particular element versions to link these element versions as part of a particular release or mile stone build. *Configuration specifications* can be created to give rules about which elements will appear in which views.

An example can be used to illustrate these concepts. A project is created to design an inventory program for a furniture retailer. As part of this project some Java classes are created to express Furniture, Chair and Table Objects. Each of these is a Java class in a Java source file. The source files are stored in Rational ClearCase. As the classes are developed entity versions of each are created and sequentially stored in the main branch for each file entity. See for example the file entity versions 0 through 6 for the Chair.java file entity in Figure 11-1 on page 318.

The first version of the inventory program is built and released into test and eventually production. The file entity versions included in this build were Table.java version 3, Furniture.java version 1 and Chair.java version 2. In order to track which entity versions went into this first release a version label is created for each of these entities. This version label is called v1. If the team wants to recreate the first release build again they know that they need to get all the files entity versions that have a version label of v1.

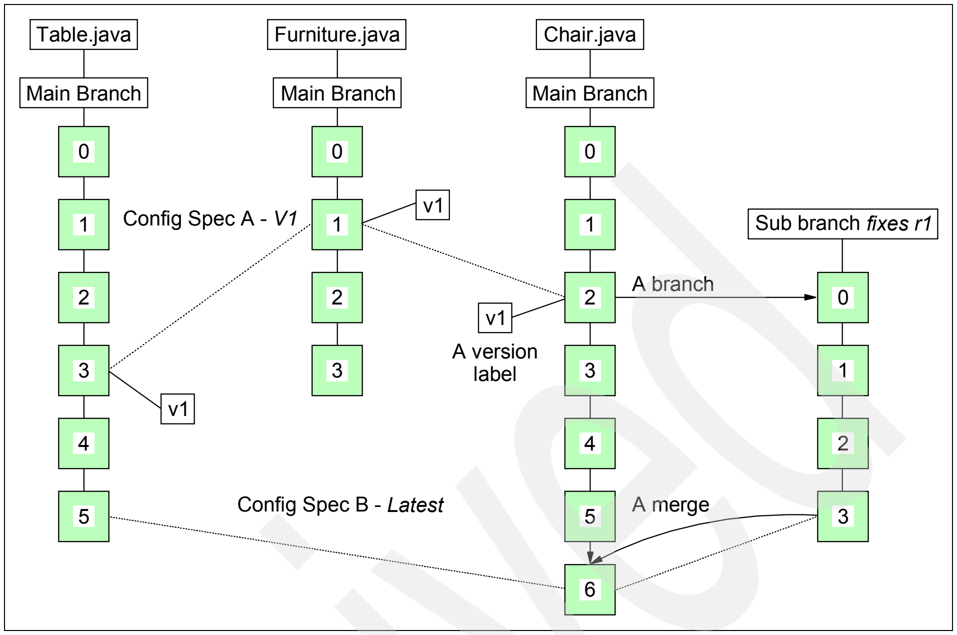


Figure 11-1 Examples of branching, merging, version labels and configuration specifications

After first release, the development team starts the next development cycle. They produce further file entity versions on the main branch of each Java file entity. File entity versions three to five are produced for the Chair.java file entity. At some point during this work various problems are found in production with release one of the application. The problems are found to be in the Chair class. Fixes need to be coded for production.

However, the versions of the Chair.java file because the first release cannot be fixed and then put into production because they are significantly different to the release 1 version. Moreover applying fixes to these versions would disrupt this second cycle of the development effort. Therefore a sub branch of the Chair.java element called fixes r1 is created. This sub branch branches off the Chair.java file entity version 2 because that was the file entity version that was in the first release. This branched version can then be included in a patch applied to production.

Various fixes and versions are added to the r1 fix sub branch during coding and during testing. The fixes in the fixes r1 sub branch are eventually merged back into the main branch of Chair.java. This ensures that the fixes are consolidated with the main branch but at a time that is appropriate for the second cycle of the classes development.

Finally, in this example, note that there are 2 Configuration Specifications created. Configuration Specification A could be used to ensure that only file entity versions that have a version label of v 1 are included in a view. This is so that a developer can check out a particular code build for release. Configuration specification B can be used to ensure that only the latest versions of all the main branches are included in the view. This is so that a developer can work on the latest versions of the code.

Note: In UCM streams are used to allow for separate lines of development. If UCM is not used then branches must be planned and managed directly by the developers and project managers. Ideally the UCM approach should be chosen as this takes full advantage of the Rational ClearCase software and other Rational software used by the Rational Unified Process

11.2.2 Concurrent Versions System

Versioning in Concurrent Versions System (CVS) is much closer to the base Rational ClearCase approach than to the UCM approach. Many of the concepts in CVS are similar to that in base Rational ClearCase but names of concepts and entities differ.

Repository

CVS stores code as directories and files in the *repository*. The repository is a specially organized directory structure that stores files. Files might have been added to the repository. This means they have been added but not committed. Uncommitted files are in the repository but not under version control. When files have been committed they are under version control. Subsequent versions of the file as they are commited are stored in the repository file system. Enough information is stored about each version to recreate that version when necessary.

Tip: Developers should commit code often but not too often.

Members of the development team should commit code often to ensure that it is:

- Inserted into CVS so that other team members have access to the latest code.
- Inserted into the back up system (which backs up CVS) regularly.

However, committing code too often can result in an unnecessary number of revisions and in the worst case scenario revisions that do not compile or that have run time bugs.

A sequence of versions of a file is known as a *branch* (just as in Rational ClearCase).

A CVS command can be issued to delete files in the repository. If a file is not committed and is deleted then it will be lost. If a version of a committed file is deleted then it is not really lost but will not be visible. If the latest file in a branch (the *head*) is deleted then the entire branch is not lost but it is placed a particular directory in the repository called the *attic*.

In short CVS manages files in such a way that once committed no version of the file is every really deleted. Unless the entire repository itself is deleted, which is why back ups of the file system are required.

Note: The Rational ClearCase equivalent of the CVS repository is a Virtual Object Base or VOB.

Versioning, branching, merging, tagging in CVS

Each file branch has a sequence of file versions. To avoid confusion between the concept of a file version and a version of a complete application the file versions are often called *revisions* and the application versions are often called *releases*.

For each file version or revision a dotted number notation is applied to that file: 1.0, 1.1, 1.2, 1.3 and so on. Sub branches continue this notation model so for example the first revision of a new sub branch that branched from the main branch revision 1.3 would be 1.3.2.1. This notation is automatically managed by CVS as you add revisions and branches so in general it does not need to be too closely understood. CVS GUI applications represent revisions and branches clearly enough for the user not to have to make sense of the numbering system. Using the CVS command line however means that the numbering system needs to be understood especially if there are many branches involved.

Sub branches can be merged back into their parent branches just as in base Rational ClearCase. Merging can result in **conflicts**. In that case the conflicting areas of code are highlighted in the uncommitted working copy of the file. This working copy of the file can be edited to resolve the conflict before this working copy is committed which will produce the merged version of the two branches.

Tip: Owners of branches should be identified. Members of the development team should be identified as branch owners. They are then the custodians of those branches.

CVS uses tags to indicate collections of file revisions that make up a significant milestone in the evolution of the code. CVS tags are the collective equivalent of version labels and configuration specifications in Rational ClearCase.

Tip: Tag names should be meaningful and follow a naming convention. Tags unlike revision numbers are the team's chance to introduce human-readable, meaningful names into the versioning system rather than just numbers. At the same time there should be an agreed naming convention to tag names so that some order is maintained and so that there is some common understanding of the meaningful names.

A common naming convention is:

```
<The>_<Applications>_<name>_<version>-<release>-<modification>-<fix>
```

For example:

```
Red_Book_Application_1-0-0-0
```

In this example, underscores are used to replace spaces between words to maintain readability while hyphens are used to substitute for full stops in the version number.

11.2.3 More tips for CVS

Other best practices for using CVS are examined here. These best practices utilize some useful features of CVS and also take into account the limitations of CVS.

Ensure all source code has a file header with CVS keywords

CVS has keywords which are words that can be added to a file with a dollar symbol ($) before and after. When the file is added to CVS the keyword is expanded or substituted with CVS information. For example, adding $Author$ gives the CVS id of the user that checked in the revision. It is best practice to create a standard comment header at the beginning of all source files. The header can include CVS keyword information and standard organizational information such as copyright information or descriptions and usage instructions.

Source code files can be stored in CVS with or without a header. However, adding a header allows for the file to be identified in terms of its revision number and other information, when it is outside CVS simply by reading the file.

Some issues with compilation and parsing can occur depending on where the header is placed at the beginning of a file. For example XML schema files need their XML version declaration before a header comment can be inserted but some experimentation of placement solves such problems.

By using the Log keyword, headers can have a detailed revison history in them which is very useful but care should be taken with this. Some source files such as interpreted files (for example HTML, Java Script or UNIX shell script files) might be better off without the history part of the CVS header. This is because the history listing can become large and also give away information that one would not want an end user to see (by examining the HTML source in his Web browser for example).

Development policy should mandate the use of headers. Developers can be encouraged/helped to add headers (at least to Java source) by using code templates in Rational Application Developer. Code templates can be set up to insert headers, including CVS keywords, and other information into source files when they are created by the IDE.

Tip: When setting up code templates in Rational Application Developer with CVS keywords, you will find that there is a clash with the use of the $ symbol to denote CVS keywords because the Rational Application Developer template system also uses the $ symbol to denote its own variables. This can be solved by using a $$ (double dollar symbol syntax) for CVS keywords. The first $ symbol escapes the second $ symbol. For example:

```
$$Log$$ rather than $Log$
```

For more information about CVS keywords lookup Keyword Substitution in the CVS manual available at:

http://www.cvshome.org/docs/manual

Communication between users is important

Users should warn other users of commit operations, CVS does have commit command options that can e-mail other users for example. It also has commands that allow users to watch files. When users watch files the file can be checked out by other users but only initially in read only mode. Users can list who is watching a file and also check out watched files in read and write mode when they have been made aware that a file is watched.

Put in place a change management process

If the development team is using CVS rather than Rational ClearCase then they do not get a prescribed change management process for CVS such as the UCM. If their organization does not have its own change management process then such a process should be created and put into place.

Further more, the process should be institutionalized in that it should be part of the processes of the organization developing the code and must involve members of the wider team such as project managers, production support personnel, business analysts and so on. Some type of requirements tracking and bug tracking infrastructure needs to be in place that then feeds into the development process and finishes with the updated code being deployed in production.

There are GNU solutions available for tracking bugs and code changes and project progress. The Apache Maven project is a possibility. For more information see:

http://maven.apache.org

However it is fair to say that the use of CVS, Maven and other GNU solutions is a patch work of solutions for project and code management. The Rational Suite® of tools is a far more integrated, end-to-end life cyle approach.

11.3 End-to-end life cycle

The WebSphere Application Server V6 environment and its connection to other Rational tools offers the developer support at every stage of the application development life cycle. Key stages in this life cycle are:

- Requirements gathering and analysis
- Prototyping
- High level design
- Low level design
- Implementation/coding/debugging
- Unit testing
- Integration testing
- Functional verification testing
- Independent testing
- Acceptance testing
- Performance testing
- Deployment
- Maintenance (including fixes, modifications, and extensions)

11.3.1 Software development life cycle key concepts

This section looks at the fundamental concepts in the software development life cycle. Trends in the industry have moved from a static waterfall model to a more dynamic iterative model. Within these models quality has to be assured by a process of verification and validation.

Waterfall model

The life cycle can be looked at as a waterfall model because the output of each stage spills into the subsequent stage. For example, when the high-level design has been created, the artifacts created during that stage, for example the design of a user interface feeds into the low level design of the modules needed to implement that user interface. This waterfall way of looking at the application development life cycle is largely non-iterative. That means that each stage is started when another stage finishes and there is little or no overlap.

Iterative model

An iterative life cycle model addresses the same stages as the waterfall model but there is some degree of overlap. Iterative means that the cycle can feed back into itself and that software grows as the life cycle is repeated.

This iterative behavior occurs at a macro and micro level. At a macro level the entire life cycle repeats itself, the maintenance stage often leads back to the requirements gathering and analysis stage. At a micro level the review of one stage might lead back to the start of the stage again or indeed back to the start of another stage.

For example, if during low-level design a flaw or inconsistency is found in the high level design then a micro HLD stage is followed again to ensure that the HLD is updated correctly. Indeed this might even feed back to a micro cycle of the requirements gathering and analysis stage if the flaw in the HLD was found to be due to vagueness in the requirements analysis.

Verification and validation

The micro iteration approach involves the concepts of verification and validation (V&V). Verification means to prove something is correct. Validation means to prove something is sound and logical. In software and systems development the two words map respectively to two key questions:

- Are we building the right system?

  This is the verification question. Is the system that we are building correct in respect to its requirements. This question can be answered by referring to the requirements analysis and design documentation. If anything is not clear or is inconsistent from these stages, then these stages require a further iteration.

- Are we building the system right?

  This is the validation question. Is the system soundly and logically constructed. Is it robust. Is it maintainable. Does it conform to standards. This question can be answered in varying degrees of formality. For example, the logic of an algorithm might be reviewed informally by a development team. At the other extreme, the logic of an algorithm might be formally proved using mathematical methods (such as the mathematical formal method language Z).

Quality Assurance

Quality assurance in software development is achieved by having some process of V&V in the development life cycle. This in turn involves some kind of cyclical review process where something is produced and then is either validated or verified. This process is iterated upon until a satisfactory level of quality is achieved.

11.3.2 The Rational Unified Process®

Rational software development tools are all built around the Rational Unified Process to a lesser or greater extent. The Rational Unified Process is an iterative software development process. It is iterative at a macro and micro level.

At the macro level phases of *Inception*, *Elaboration*, *Construction* and *Transition* can be identified in the process. These phases are basically periods of initial planning, more detailed planning, implementation and finalizing and moving on to the next project cycle. The next cycle will repeat these phases. At the micro level each phase can go through several iterations of itself. For example during a construction phase: coding, testing and re-coding can take place a number of times. Figure 11-2 gives an overview of the Rational Unified Process.

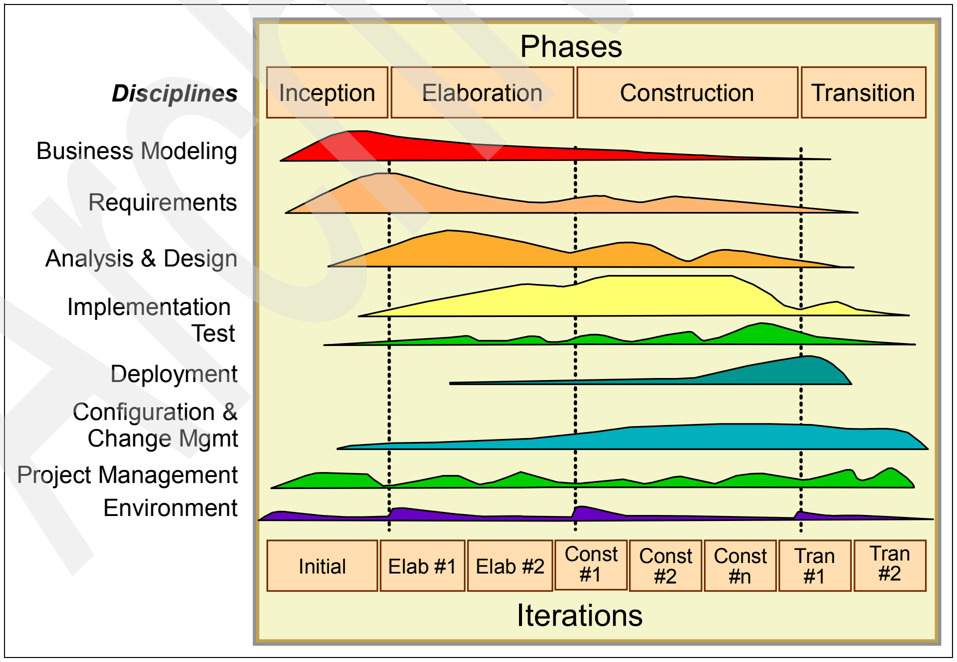


Figure 11-2 Rational Unified Process overview

Also shown in Figure 11-2 on page 325, the Rational Unified Process identifies a number of disciplines that are practiced during the various phases. These disciplines are practiced during all phases but the amount of activity in each phase varies. Clearly the requirements discipline will be more active during the earlier inception and elaboration phases for example.

The Rational Unified Process maps disciplines to roles. There are many roles but the roles break down in to 4 basic sets of roles: *Analysts*, *Developers*, *Testers*, *Managers*. Members of the team can take on more than one role. More than one team member can have the same role. Each role might require the practice of more than one discipline. Table 11-2 shows the basic mappings between roles and disciplines.

Table 11-2 Disciplines mapped to Rational Unified Process role sets

	Analysts	**Developers**	**Testers**	**Managers**
Business Modeling	X			
Requirements	X			
Test	X	X	X	
Analysis & Design		X		
Project Management				X
Configuration and Change Management				X
Process Configuration and Project Environment				X
Deployment				X

Note: Each discipline and each role in Table 11-2 has many sub roles and sub disciplines. The table is simply an overview.

The Rational Unified Process can be followed without using Rational Software, it is just a process specification after all. However, the Rational Unified Process provides specific guidance (called Tool Mentors) on how to use Rational Software when following the process. The disciplines identified in the Rational

Unified Process such as requirements analysis, design or testing map to specific pieces of Rational software and artifacts that this software generates. The RUP is a process that can be ‘bought into’ as much or as little as is required.

For information about Rational Software offerings and how they map to phases in the life cycle, see Chapter 6, “Development tools” on page 173. For more information about the Rational Unified Process, see:

http://www.ibm.com/software/awdtools/rup

11.4 General development best practices

Any development effort would benefit from the following best practices. Emphasis is placed on an iterative and object oriented approach as these are the prevailing trends in contemporary project management and development.

11.4.1 Iterative Development

During development there are always unknowns. These unknowns can take the form of the use of new technology, the use of new development staff or the use of new ideas to solve problems. These unknowns then take the form of risks. The earlier these risks are exposed; the earlier they can be mitigated.

Developing iteratively means to go through a number of sub development cycles in turn in order to expose risks early. This is in contrast to the waterfall model. In a waterfall development model, because testing occurs late in the cycle, risks (such as the use of new technology and its unknowns) are not exposed until the components that use that technology are built. Or worse, they might not be exposed until those components are unit, integration and acceptance tested.

Iterative development steps through sub iterations of the entire development process. This means that fundamental parts of the application are prescribed, designed, implemented, tested and deployed in very controlled iterative cycles.

The key word here is controlled. The objective of the iteration has to be very clear and self contained. Discipline in project management and software development through out the micro iteration have to be as high as it would be in the entire macro iteration.

For example, a project called RedbookApplication might require the creation of a Web front end using Struts technology, an EJB back end including the use of Java Messaging Service technology and an EJB client application. The front- and back-end layers might require many interface and back end components. However, a first iteration might simply design, code and test a single Web interface component, a single back end component and a single EJB client component. The Web interface would make use of Struts in this first iteration and the back end component would make use of Java Messaging Service technology at this early point.

This first iteration is not implementing all the Web interface, backed EJB and EJB clients needed. But what it is doing is creating a single example of each. What it is also doing is creating parts of the application that will be needed rather than prototyping with functionally irrelevant example code. Finally, and perhaps most importantly, it is implementing and testing the use of the specific technologies such as Struts and Java Messaging Service which can be new to some or all members of the development team.

Further more, there can be novel ideas and or new design patterns that are being used. An example of these should be included also in this first iteration.

It is important here that the iteration is producing functionality that is needed in the final application. The iterative approach is similar in some ways to prototyping. Prototyping is the practice of building an application to prove theories from the design stage. It is also possible to show a prototype to end users and or customers in order to ask: Is this the sort of thing you wanted? It is commonly a stage somewhere between requirements gathering, design and coding stages in the waterfall model.

The dangers with prototyping are that while prototyping produces an application that tests theoretical and new unknown areas it is not focused on necessarily producing functionality required in the end product, it is often discarded which in some ways can be seen as a waste of development effort. On the other hand, and in the worst case, if the prototype is concerned with producing final functionality, it might do so in a way that is not robust or well designed, just to get something ‘up and running’. This poor design and lack of robustness can then find its way into the final product. It has been known for customers, when shown a prototype, to think it is so good (and cheap, because they have something that 'works' and in a short period of time) that they want to use that as the final product.

The iterative approach takes the benefits of prototyping but it avoids its pit falls. The deliverables of the iteration are deliverables for the final product. By following the iterative approach risks are identified early and mitigated. There are other benefits in developing iteratively beyond risk mitigation:

Absorbing changing requirements

The iterative approach also allows changing requirements to be absorbed. Requirements will change. Life is not static and even during a short development project, there is enough time for customers or end users to change their minds. These changes can have good reasons behind them. The environment on which the application is to be used can change. The market to which the business is responding can change. The experience of the end users or domain experts can change (often as a result of focusing on the requirements for the application). These changes are not necessarily to be discouraged as they can lead to genuine and necessary improvements in the original requirements and designs. If an iterative approach is being followed then these changing requirements can be absorbed more easily during the current iteration.

Quality

Quality improves through iterative development. By definition, the application goes through a number design, coding and testing cycle revisions this enhances the application's robustness, performance and maintainability.

Learning

Members of the team learn lessons. They can apply the knowledge in the same macro development cycle or project. If a team member needs specific training these needs are identified earlier on and the knowledge learnt can be applied in the same project. Team members also learn about the development process itself. The process or the team member's practice of the process can be tweaked and improved upon.

Reuse

Reusable components, or common components might turn out to be not so generic in practice. Likewise, some components, which were thought not to be generic, can emerge as generic as other uses become apparent. These design considerations might not emerge until the project is well under way. Also the use of third party technology (for example a GNU API such as Apache Xerces XML API) can be evaluated. Sometimes third party technology can turn out to be inappropriate and another alternative needs to be sought, sometimes the technology can have additional, unexpected benefits that one would like to take advantage of too. The iterative approach makes these adjustments possible.

In summary, software development projects are learning processes no matter how well planned. The iterative model allows for controlled and sub iterations of the entire life cycle with very clear goals for each sub iteration. This iterative development model allows the team to take advantage of new knowledge earlier on. Hind-sight is taken advantage of before the macro iteration of the project life cycle ends.

11.4.2 Requirements Definition

Capturing requirements and managing changes to those requirements is fundamental to a successful project. Clear requirements and a process for managing changes to those requirements gives developers a chance to design and implement the right system. If requirements are vague and the process for managing changes to requirements is poor or non-existent then at best the deliverables are going to be late and at worst they are going to be late and incorrect. The use of use-cases is best practice when modeling requirements. These use cases can directly be used to design components and devise test cases.

A use case is a scenario in the application that has actors and outcomes. It is a model of the permutations of events that can occur or be allowed to occur during that scenario. Use case diagrams are a part of the UML and are a standard way of expressing these scenarios. It is possible to write use cases without creating use case diagrams. It is highly recommended that use cases are identified and created during the requirements stage to achieve a successful development project outcome.

11.4.3 Object-oriented approach to design and programming

This seems an obvious statement, given the use of contemporary object-oriented languages, but this is an important point to make:

> It is all too possible, but undesirable, to use object-oriented languages in a procedural way (just as it is possible to write object-oriented programs using procedural languages).

Developers using object-oriented languages do not necessarily follow best practices or think in a component-based manner. Developers must think in a modular way during design, coding, and test. The application is a module made up of smaller components working together. The key concepts that should be considered are *cohesion* and *coupling*. Developers should be aiming for high cohesion and low coupling in modules.

High cohesion means that areas of common data and behavior are encapsulated in a single object or related set of objects. This approach groups behavior and data into manageable and reusable components.

Low coupling means that the interfaces of each component that allow the components to interact should be kept as simple as possible and calls to interfaces should be kept as few as possible.

Tip: The success of the approaches is cumulative. If the right decisions are made in making modules and groups of modules cohesive, then the goal of low coupling is a lot easier to achieve.

If you do not understand this concept, then consider the placing of a fire hose on an ambulance rather than on a fire truck. No doubt the ambulance sometimes is in the same place as the fire truck, and the firefighters can use the hose. However, clearly it would be better if the hose were part of the fire truck. This might seem like common sense, but in the abstract world of software development, such clarity can sometimes be harder to come by. When in doubt, applying first principles such as the laws of high cohesion and low coupling can help clarify the situation.

Good object-oriented approaches and thinking lead to elegance, reusability, and easier maintenance. Good approaches can be surprisingly rare among developers using object-oriented languages and technologies. Use of design patterns and IDE tools can help guide the developer into good practices but understanding object-oriented design principles is fundamental.

11.4.4 Modeling languages

UML is a visual way of expressing concepts in software development. Use cases and class architectures for example can be expressed unambiguously. Unified Modeling Language does not have to be used but developers need to have a way of communicating design ideas clearly and without extraneous details. UML class diagrams for example are one view of the OO component architecture. They show the entities that the developer wants to show and do not show those that would confuse issues. Rational Rose and Rational Application Developer both have UML diagram capabilities. It is recommended that developers use this functionality.

Note: UML can be used passively, especially with class diagrams. This is important because developers new to UML will find it far easier to read diagrams than to create diagrams. Components such as classes and packages can be created in Rational Application Developer (and other Rational Tools such as Rational Rose). These components can then be dropped onto a diagram panes and the class diagram UML is automatically generated. These diagrams can then be exported to design documents. The auto-generation of UML means that even developers that are relatively new to UML can use it. Developers can passively read the diagrams rather than actively produce the diagrams.

11.4.5 Regular Reviews and check points

During development, design and code reviews are invaluable. Without them quality suffers. Designs must be reviewed against requirements. Code must be reviewed against designs. Code must also be reviewed for other quality standards. We will look at some of these standards later in the section on Java Development best practice. Design and code reviews can have varying levels of formality. What is key is that the review process should:

- Schedule time for developers to collectively look at specific design or code modules.
- Record findings formally so that there is a permanent record of what needs to be changed.
- Schedule time for developers to make the changes to the design and code artifacts.

Reviews should be held regularly as per the iterative approach. This keeps developers 'on the same page' and corrects problems early on. Regular reviews also keep reviews fresh by reviewing smaller chunks of code. Reviewing all the code at the end of implementation could be so monotonous, given the amount of code to review, that the quality of the inspection is significantly reduced.

11.5 Java development best practices

Java will naturally be the development language of choice in an Enterprise Java Environment. Java Standard Edition is the core platform for all Java development projects including Enterprise Java projects. Here we look at best practices that apply to all projects using Java Standard Edition. Some of these practices are also good practice for any development project.

11.5.1 Code documentation

Undocumented code is next to useless. Documentation best practice is fundamental to the design, implementation and maintenance phases. If code is not properly documented then code maintenance in subsequent iterations can be at best problematic and at worst impractical. Code maintenance has many sub areas. Extending applications, scaling applications and fixing applications are all forms of maintenance. Without good documentation these tasks are much harder to complete.

JavaDoc gives developers a contract

JavaDoc is used to document classes, instance variables and methods. It has very clearly defined annotations that can aid the development process considerably. For example, the method JavaDoc comments have annotations to describe input parameter values, output values such as return values and exceptions and under what circumstances these are produced. As such the method JavaDoc comment, if properly implemented, represents the method contract.

Given that JavaDoc gives such a precise way of documenting the modules in an application, JavaDoc documentation of code is not something that should be left to chance. Indeed, JavaDoc comments are artifacts in the development process that should be written at design time. There are various ways of doing this.

A low-level design document might simply describe all the modules that are to implement the system. The JavaDoc comments can then be written into the low-level design document. When it comes to coding. The low-level design then forms the template for developers who have been allocated particular modules. They can cut and paste the JavaDoc comments into their source files.

A design tool such as Rational Rose can be used to design components. As components are created, they can have their JavaDoc class, instance variable and method comments filled in. When the design is complete design documentation can be generated in the form of UML diagrams via automated design documentation tools such as Rational SoDA. Additionally Java class shells can be generated. That is Java source code can be generated from the Rational Rose model including all the class, instance variables and method signatures along with all the JavaDoc comments. These classes can then be allocated to each developer for implementation.

Whether using design documents with JavaDoc comments in them or more sophisticated design tools such as Rational Rose that generate code shells; the outcome of this approach is that the JavaDoc comments which form contracts are placed in front of the developer before they start coding. This keeps the contract for the module in front of the developer as he codes the module. By sticking to the class and method contracts the developer is more likely to achieve the desired outcome: a module that is going to pass Unit Test first time around.

Local comments best practice

During coding, code should be commented where necessary. The comments we are talking about in this case are not JavaDoc comments. They are local comments written at some point in an algorithm inside a method body. The general best practice rule of thumb for writing effective comments is that the comments should explain 'why' not 'what'. Well-designed, well-written code is itself a documentation of what is happening. If code is so poorly designed or esoterically written that it is not clear what is happening then the developer should consider being more explicit in their coding style or re-factoring the code. Why something is being done is not always so clear locally. Variables and method calls often come from separate modules. Non-procedural languages do not put everything in front of the developer at the same time. Explaining why something is being done in a particular way is what local comments should be used for.

Save time in code reviews

Good documentation of code saves time during code reviews. It allows other developers to read code more easily. It also cuts down on findings from reviews. One of the major, and unnecessary, types of finding in code review is the need for more accurate JavaDoc and local comment blocks. If those blocks are found at all.

11.5.2 Unit testing and JUnit

Unit testing is the testing of discrete modules to ensure that their interfaces take the correct inputs and give the correct outputs on the basis of those inputs. The WebSphere environment comes with IDEs that support JUnit for unit testing. JUnit is available as a separately down-loadable install that can be run out side an IDE. It is therefore a resource that is worth using and understanding. The following sections describe some JUnit best practices.

Make JUnit tests simple

JUnit tests should be simple in that they should test a discrete area of functionality. This means that areas of functionality can be tested in isolation. If larger areas of functionality need to be tested then test cases can be written to run groups of test cases. Java modules are designed discretely and their test cases should reflect this.

Make JUnit tests generic

The whole point of having a set of unit test modules is that they can be run at any time to validate the system and all of its sub components. This will not be the case if the tests are not generic. Generic means that the test is not dependent on:

- Any particular external conditions or state.
- Operating system features such as file system idiosyncrasies.
- Outcomes or data from other tests.
- Local time zones.
- External configuration files.

JUnit packaging and test data

It is a useful practice to package JUnit tests separately to the modules that they test. When deploying compiled code it might or might not be desirable to export JUnit classes at the same time as exporting application implementation classes.

Test Data can be packaged in files in the JUnit packages. The java.lang.Class.getResource() method call can be used to find files containing test data resource files that are stored in the package. This also avoids absolute file paths.

Rational Application Developer has wizards for developing JUnit tests. In this latest incarnation of the IBM IDE there are also wizards for Test projects that use test templates for testing different types of modules. Templates for simple method testing, interface testing and EJB testing are all supported. In each case, JUnit test cases are generated and can be tweaked by the developer when generated. As these wizards organize the code in test projects the organizational concerns that might affect the developer are less of a burden.

For further information about JUnit, refer to the Rational Application Developer product documentation or see:

`http://www.junit.org`

11.6 Enterprise Java development best practices

This section looks at best practices specifically in the Enterprise Java field and also with some WebSphere Application Server V6 specific concerns.

11.6.1 Design patterns, design patterns, design patterns

Commonly used solutions in application architectures fall in to categories. These are called design patterns. The Enterprise Java arena uses these design patterns and also has some Enterprise Java Specific design patterns too. The following are generic design patterns that are not specific to J2EE nor to the Java language. However, they are design patterns that are used very extensively in J2EE.

Model, view, controller

The model, view, controller (MVC) pattern originates from tools developed for the Small Talk programming language. The MVC pattern follows good object-oriented design principles. That is, it promotes high cohesion and low coupling. Most applications have interfaces and business logic.

Interfaces can be human user interfaces such as command lines and GUIs. Interfaces can also be middleware interfaces (which are essential application to application interfaces). Business logic and back-ends deal with resources interaction such as interaction with databases.

In order to make applications highly maintainable (which means they are extensible and easy to understand and modify regardless of future requirements) it is desirable to keep the coupling of interfaces and back ends as low as possible. The MVC pattern prescribes that the interface and the back end are entirely separate and that all communication is done via a controller layer. This controller layer introduces indirection into the application. This means that any type of interface (a GUI, a command line or an application interface) can be coded to talk to the controller layer. And in addition any back end application layer can talk to the controller layer too.

MVC is fundamental to Enterprise Java. In the simplest form JSPs and EJB client applications represent the View. Servlet and Session Façade layers represent the controller. Entity and message-driven beans represent the Model.

However, as with object-oriented approaches and standard Java, just because a developer is given the tools to create object-oriented or MVC architectures does not mean that the developer will create object-oriented or MVC architectures. It is possible for example to use a JSP to call an Entity EJB directly. This is logically correct but might not be the best design solution in terms of keeping the interfaces between the back and front ends simple. During development developers should keep the MVC goal in mind and be aware that there is every danger of breaking this pattern despite using J2EE technologies.

Façades

The façade design pattern offers a more simplified interface by encapsulating a more complex interface or collection of interfaces.

Singletons

Singletons are classes that return an instance of them selves to calling classes. They only return a new instance of them selves to calling classes if the single instance of themselves that they maintain has not been created yet. The advantage of this is the avoidance of static method calls. Static method calls are not a very clean OO approach. More over ensuring that only one instance of a Class exists at any one time means all users of the object, reference that object rather than maintaining there own instance. Separate instances represent a memory overhead.

However, a word of caution on singletons. Singletons are only desirable if the object is totally generic in all situations for all calling classes. If a singleton object might need to be specific at some point in the future then the singleton pattern should not be selected.

Factories

The factory design pattern is where an object has a dedicated purpose. The purpose is to produce instances of a particular object class. The factory object is the single point of access for obtaining an instance of the object. The factory design pattern is often used in conjunction with the singleton pattern that ensures that only one instance of the factory object every exists in memory.

These generic design patterns are used extensively in J2EE and the following sections on J2EE performance make use of these patterns and examine some J2EE specific design patterns too.

11.6.2 J2EE performance issues and best practice

The MVC pattern is the overriding design pattern in the J2EE application architecture. The MVC is made up of cohesive View, Model and Controller layers of areas of functionality that communicate with one another through loosely coupled interfaces. Performance can be highly affected by the implementation of these layers and the ways that these layers communicate with one another. In this section we look at best practice when designing and implementing these layers for high performance in a J2EE architecture.

View layer: JSPs, Struts, JSF

JavaServer Pages are HTML presentation layer pages with special embedded tags that are processed on the server side within special Servlets. The Servlets are automatically generated during deployment. An increasingly sophisticated library of standard tags is available which is over time reducing the amount of Java code scriptlets that have to be used. The use of pure JSPs (rather than using Struts or JavaServer Faces (JSF) gives the best performance. However, Struts and JSF architectures have other benefits which should be considered.

Struts (which is an apache org sub project) shields the developer from MVC architecture considerations and allows for better maintainability. JSF gives the developer a wide variety of specialized GUI components to choose from. Struts and JSF are now more comprehensively supported in WebSphere and Rational Application Developer and Rational Web Developer than ever before.

JSPs and Servlets

JSP and Servlet performance related best practices include:

- Minimizing the include JSP tag statements as each is a separate servlet.
- Use the `usebean` JSP tag only to access existing beans not to instantiate new instances.
- If a session object is not required in the JSP use the <%page “session=false” %> directive to avoid the default creation of a session object.
- Avoid the use of the SingleThreadModel for Servlets or client requests will be sequential rather than running in parallel.
- Place expensive one time initialisation calls in the place Servlet init method rather than the doGet and doPost method calls.

Struts

Struts is a framework that enforces MVC. Prior to Struts developers used combinations of JSPs and Servlets to create the View and Controller layers of an application. Struts took best practices that emerged from the JSP and Servlet architectures and created an architecture that introduces an Action model where JSP form data is submitted to a controller ActionServlet.

Struts uses JSP tag libraries extensively. WebSphere Application Server V6 now comes with caching features specific to Struts architectures to reduce the overhead of tag library references.

Where possible JSP rather than XSLT should be used to render the view layer as XSLT (which is the transformation of XML to HTML pages) is expensive.

Form Beans should not be passed to the business logic layer. The layers should be completely separate. Passing form beans to the back end would mean that the back end is not independent from the front end in terms of the objects it needs to know about.

JavaServer Faces

JavaServer Faces split the user interfaces presentation and processing cleanly which allows for easier construction and attachment of multiple user interfaces. The components available are varied and sophisticated which increases the possibility of what GUI components can be used in the front end. This richness of features and further level of indirection between the interface and processing comes at a cost. JSF can be relatively poor in performance when compared with JSP and Struts. The decision to use the technology has to weigh the merits of a sophisticated and extensible interface against the performance costs. Consideration of how much data a JSF component can display and trying to keep that to a minimum can ameliorate the performance cost.

Controller layer: Servlets and Session Beans

Servlets can be used to implement the controller layer which communicates with the presentation layer. The use of Struts and JSF generates this layer as part of those architectures which is one of the virtues of using those technologies. The following are best practices when implementing a controller layer:

Keep presentation layer and business logic layer functionality out of the controller layer. For example the controller layer should not generate HTML nor should it process business logic algorithms.

Place common functionality in the controller layer into a common super class.

Each controller component should implement a specific, discrete task. For example a separate servlet should be create for logging, in logging out, uploading data etc. Putting all controller tasks into a single Servlet can lead to maintenance and configuration problems because the law of high cohesion has not be followed. That is each area of functionality should be split into a discrete module: in this case separate Servlets.

Maintaining state

In general using the HTTP session object is used to maintain state specific to Web clients. Best practices for HTTP session objects are:

- Minimize the size of sessions objects by storing only essential data for the application there.
- Invalidate unused HTTP sessions explicitly rather than relying on session time outs.
- HTTP Session should not be cached by the application or they will not be reclaimed when the session expires or is invalidated, this can lead to memory leaks.

Stateful Session Beans can be used to maintain state information too. Especially when the front end is not Web based. Such beans are not shared by multiple clients nor are they pooled. Stateful Session Beans exist until they are explicitly removed or until their session expires. There fore when using Stateful Session Beans remove Stateful Session Beans when the client is finished with them. Otherwise, they will continue to exist in memory and or be passivated to disk which represent needless overheads if the bean is no longer required. The bean can be explicitly removed by invoking its `remove` method.

Model Layer: EJB performance issues and best practice

The model layer in an Enterprise Java application is where the real work is done. It is here that business rules and processing is done and where data is brought in and out of persistent storage.

When Not To Use an EJB

EJBs are objects that can be distributed, support transactions, can be pooled and support the J2EE security model for containers. If an object does not need to make use of these features then the object should not be an EJB. EJBs are expensive resources in terms of communications and memory usage and should be used only when their advantages can be exploited by the application, not as a matter of course.

EJB interfaces

EJBs can be referenced remotely or can be referenced locally. Evaluate early on if a remote, distributed call is really required. If not then use local interfaces this makes an application far more efficient as the call is done by reference rather than by remote method invocation. This has been true because the EJB 2.0 specification. Before then remote calls were used for all communications. Clearly the use of local interfaces will limit the deployment of the application to a cluster.

Encapsulate data passed between distributed objects

Distributed method calls are expensive. There fore when making a distributed method call it is desirable to get as much data to and from the method as possible. Returning an object containing a collection of data, rather than a single data item is more efficient. If both options are required under different circumstances then both methods can be specified but the use of the collection access method should be favored. This is far less network intensive than making a number of method calls for individual items. A single method parameter object that encapsulates a number of items that would otherwise be sent as separate parameters is also more efficient.

Use Container Managed Transactions

Use the container managed transactions functionality rather than developing your own transaction management. The container managed transactions execution is almost always more efficient and far easier to maintain.

J2EE Design Patterns in the Model Layer

Here are some J2EE Design Patterns that are useful for the business logic layer.

Session Façades

A Session Façade pattern is constructed from a stateless Session EJB accessing one or more Entity EJBs. The Session Façade is based on the Façade design pattern (which is not J2EE specific). A Façade simplifies the interface of a more complex underlying system. In a Session Façade the complexity of finding and calling a number of Entity beans is simplified into one call to a method on a Session Bean. This pattern in turn reduces complexity in client layers.

This pattern also reduces remote method invocation overheads because the calls from the view layer to the controller layer are done locally.

It also reduces transaction handling overheads because it means that the view objects are not accessing the Entity beans directly. When view object access entity beans directly separate transactions are created which is an extra overhead.

Data Access

EJBs to access databases and other resources are not always necessary and if this is the case the memory and network overheads of using an EJB can be avoided. Data Access patterns are façades objects that provide a simple interface to client modules and implement the connection and retrieval code internally. A straight JDBC connection might be an example.

Service Locator

A single object can be used to do JNDI lookups for resources such as EJBs and Java Messaging resources. Delegating this task to one object that is then called by other objects is far better practice than all objects doing the same JNDI call to look up objects.

EJBHomeFactory

This is a composite design pattern that uses other design patterns. Its purpose is to reduce the overheads when looking up home interface references for EJBs. This design pattern is similar to the Service Locator pattern. It also uses the singleton pattern described earlier (that is it maintains a single instance of itself for use by other modules). It also uses the factory design pattern (that is, its soul purpose in life is to produce instances of a particular object type for other modules).

Avoiding memory leaks

Java manages memory therefore you don't get memory leaks in Java right? Wrong! Resource connections should be cleaned up carefully. Some tips for avoiding memory leaks are as follows:

- Use `finally` rather than exception blocks to clean up after resources to ensure it is done (complex `catch` exception handling misses some cases)
- When using a singleton pattern set references to resources to null explicitly or they might not be released for garbage collection

For more information about development best practice for performance refer to *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392.

For more information about J2EE design patterns, see:

http://www.theserverside.com/patterns

11.6.3 WebSphere Application Server clients

When you develop a client application using and adhering to the J2EE platform, you can put the client application code from one J2EE platform implementation to another. The client application package can require redeployment using each J2EE platform deployment tool, but the code that comprises the client application remains the same.

The Application Client run time supplies a container that provides access to system services for the client application code. The client application code must contain a main method. The Application Client run time invokes this main method after the environment initializes and runs until the Java virtual machine code terminates.

The J2EE platform supports the Application Client use of nicknames or short names, defined within the client application deployment descriptor. These deployment descriptors identify enterprise beans or local resources (JDBC, Java Message Service, JavaMail, and URL APIs) for simplified resolution through JNDI. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the client application code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the Application Client can require redeployment.

The Application Client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The Application Client run time also provides support for security authentication to enterprise beans and local resources.

The Application Client uses the Java Remote Method Invocation-Internet InterORB Protocol (RMI-IIOP). Using this protocol enables the client application to access enterprise bean references and to use Common Object Request Broker Architecture (CORBA) services provided by the J2EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the J2EE and CORBA environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

Application client functions

Table 11-3 identifies the available functions in the different types of clients.

Table 11-3 Application client functions

Available functions	**ActiveX client**	**Applet client**	**J2EE client**	**Pluggable client**	**Thin client**
Provides all the benefits of a J2EE platform	Yes	No	Yes	No	No
Portable across all J2EE platforms	No	No	Yes	No	No
Provides the necessary run-time support for communication between a client and a server	Yes	Yes	Yes	Yes	Yes
Supports the use of nicknames in the deployment descriptor files. Note: Although you can edit deployment descriptor files, do not use the administrative console to modify them.	Yes	No	Yes	No	No
Supports use of the RMI-IIOP protocol	Yes	Yes	Yes	Yes	Yes
Browser-based application	No	Yes	No	No	No
Enables development of client applications that can access enterprise bean references and CORBA object references	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run-time environment	Yes	No	Yes	No	No
Supports security authentication to enterprise beans	Yes	Limited	Yes	Yes	Yes
Supports security authentication to local resources	Yes	No	Yes	No	No
Requires distribution of application to client machines	Yes	No	Yes	Yes	Yes
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code	Yes	No	No	No	No
Provides a lightweight client suitable for download	No	Yes	No	Yes	Yes

Available functions	**ActiveX client**	**Applet client**	**J2EE client**	**Pluggable client**	**Thin client**
Enables access JNDI APIs for enterprise bean resolution	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment	No	No	No	Yes	No
Supports CORBA services (using CORBA services can render the application client code nonportable)	No	No	Yes	No	No

12

Planning for application deployment

This chapter provides an overview of the deployment options and considerations in a WebSphere Application Server V6 environment. Deployment means the configuration, installation, and update of applications in the WebSphere Application Server. This chapter contains the following sections:

- Planning for the deployment of applications
- WebSphere Rapid Deployment
- Annotation-based programming
- Deployment automation
- Best practices for deployment

12.1 Planning for the deployment of applications

This chapter continues the analysis from the development point of view. Chapter 6, “Development tools” on page 173 discussed development tools. Chapter 11, “Planning for application development” on page 305 discussed general development considerations. Now, this chapter looks at the options that are available for deploying the developed applications to the WebSphere Application Server environment and why some of these options are better than others in particular circumstances.

This section provides a brief reminder on the tools that are available for deployment and gives some general considerations for deployment in the different topologies.

12.1.1 Deployment tools available

The WebSphere Application Server V6 environment provides the Application Server Toolkit for deploying applications to servers. The Application Server Toolkit comes as part of WebSphere Application Server V6, Rational Web Developer, and Rational Application Developer.

In addition to this toolkit, WebSphere Application Server V6 also has other deployment mechanisms. There is the administrative console, which is a Web-based GUI that is implemented by a special Java enterprise application. This is installed on Deployment Manager nodes and stand-alone WebSphere Application Server nodes. Also there are the command line scripting tools as **`wsadmin`** and **`Ant`**.

12.1.2 Topology considerations

Chapter 9, “Topologies” on page 247 gives a detailed analysis of various topology options in a WebSphere Application Server V6 environment. Successful planning for deployment entails consideration of the various WebSphere entities in various parts of the network and how this affects the ability to configure, install, and update applications on these entities.

Deployment client and the Deployment Manager

A deployment client is whatever client is being used. This would be the Web browser (in the case of the administrative console) and the command line executable, such as **`Ant`** or **`wsadmin`** (in the case of command line administration). To deploy an application, there must be network access between the deployment client and the Deployment Manager node (in a WebSphere Application Server Network Deployment V6 environment) because this is the single point of administration for all nodes. In a stand-alone environment there must be network access to the single application server. Development and test environments cannot be as complex or have as many restrictions on access. Production environments, with their more restrictive tiers and firewalls can present more issues. For remote Web and command line deployment, access to the administrative console and SOAP ports, respectively, is required. For a more detailed discussion of these ports see 10.4, “TCP/IP ports assignment” on page 296.

Development environment and the Agent Controller

In a development environment, it is possible to profile applications on remote servers for performance purposes. Rational Application Developer can deploy an application to a remote node and run, debug and profile the application on that node. In the case of profiling, an Agent Controller installation is required on the remote application server node. The Agent Controller software comes with Rational Application Developer. (Not to be confused with a Node Agent process; it is not the same thing.) Access to the Agent Controller is required via a designated port on the remote machine. This port is 10002 by default. Clearly there must be port access between a developer’s machine running Rational Application Developer and the machine running the Agent Controller.

Tip: You can change the default port for the Agent Controller. Changes must be made in a configuration file under the Agent Controller and in a preferences GUI in the Rational Application Developer.

Web server plug-in considerations

Web servers are another consideration when planning for deployment, because they are often in a separate network tier from the WebSphere Application Server layer (especially in a production environment). After the installation of an application in the application server environment, the Web server plug-in (which tells the Web server which HTTP virtual hosts and context roots to pass on to the application server layer) might need to be updated. For example, when a new Web application is deployed with a new virtual host or context root, the plug-in needs updating.

If the Web servers are in a separate network layer, this might be a problem. There are three possible administration setup models for a Web server on a separate node:

- Managed-node
- Unmanaged-node
- Unmanaged-node (special case)

In the managed-node and the unmanaged-node (special case) models, it is possible to update the Web plug-in configuration under the Web server via the

Deployment Manager. This is because in the managed-node model there is a node agent running on the Web server node and in the unmanaged-node (special case) model (only for IBM HTTP Server V6) there is the Administrative Server running. In the unmanaged-node model, as there is no administration process running on the Web server node, it is not possible to use the Deployment Manager to update the plug-in configuration on the Web server. In such a case the plug-in configuration file has to be copied manually from the application server machine to the Web server machine.

In a production environment, the unmanaged-node model might be obligatory because of network security. That is, keeping to a minimum the number of connections between layers. In a development environment the network security constraints can be less stringent.

Clearly though, you need to take into account the above issues with the Web server plug-in configuration when designing the environment. What options you choose is often a trade off between security and productivity. A development environment where the frequency of application installation and updates is likely to be high would be more productive if the plug-in can be updated via the Deployment Manager processes.

12.2 WebSphere Rapid Deployment

The WebSphere Application Server V6 environment provides new functionality to make application development and deployment quicker and easier. The WebSphere Rapid Deployment functionality comprises of two key concepts:

- Annotation-based programming

  A new feature implemented in the Rational Application Developer, Rational Web Developer IDEs, and the Application Server Toolkit.

- Deployment Automation

  A WebSphere Application Server process that checks for updates to application artifacts in a specific directory and automatically installs those updates.

12.3 Annotation-based programming

A feature of WebSphere Application Server V6 is the support of annotation-based programming.

Metadata

To understand annotation-based programming, you first need to understand metadata. Metadata is data about data. For example, a Java servlet is written in a Java source file but information about the servlet (such as its start up parameters) is stored in the Web deployment descriptor XML file. The XML file stores metadata about the servlet.

This example can be extended further. The deployment descriptor XML file references an XML Schema document which contains rules on which XML elements and attributes can be used in the deployment descriptor document. The schema document stores metadata about the deployment descriptor.

Annotation-based programming definition

Annotation-based programming is where a Java source file contains special tag syntax which can be parsed to create other external, metadata artifacts at compilation, deployment or runtime.

In the servlet example mentioned above, tags are written into the JavaDoc class comment block of the servlet source file specifying the servlet parameters. When saved, the IDE will take care of updating the web.xml (Web Deployment Descriptor) metadata file. This update allow for the specification of all data concerning the Java source file in that file itself. The developer can focus on a single file or artifact rather than on other metadata artifacts, thus increasing productivity and simplifying deployment.

12.3.1 A past, present, and future trend

Annotation-based programming in Java has been evolving for some time, and there are some clear directions on where it is going in the future. The following gives an overview of this evolution:

- JavaDoc

  JavaDoc was the only tool in the core Java platform (prior to Java 1.5) to process annotated Java code and generate metadata. The metadata generated was only HTML documentation of the source code. Document blocks could be added to the Class, instance variables and methods allowing for the comments to be added with scope as to which parts of the code they were pertinent to.

- XDoclet

  XDoclet is an open source code generation engine that takes a Java source file (annotated with special JavaDoc tags) and generates metadata files for the developer based on those tags. WebSphere Application Server V6 uses XDoclet tag syntax for J2EE 1.3 and will adopt J2EE 1.4 tags when XDoclet 2 is released. WebSphere Application Server V6 does not use the XDoclet engine, it has its own annotation processor. Additionally, WebSphere Application Server V6 has its own proprietary tags specific to WebSphere development. You can find further details about XDoclet at:

  `http://xdoclet.sourceforge.net/xdoclet/`

- Java Specification Request 175

  The Java Community Process (JCP) organization produces Java Specification Requests (JSRs) for the future direction of the Java platform. The *JSR-175: A Metadata Facility for the Java Programming Language* is the specification for incorporating metadata generation into the core Java language. Thus, supporting metadata by annotation-based programming is something that most likely will be used increasingly in all areas of Java programming in the future. You can find further details about JCP at:

  `http://www.jcp.org`

- J2SE 5 Annotation Support

  The WebSphere Application Server V6 environment supports J2SE 1.4. The next version of Java 2 Platform, Standard Edition is 5 (also know as Tiger or Java 1.5). J2SE 5 supports annotations as part of the core API and compiler. It implements the Java Specification Request 175. You can find further details about J2SE 5 specification at:

  `http://java.sun.com/j2se/1.5.0`

- Java Specification Request 220

  JSR 220: Enterprise JavaBeans 3.0 is the JCP specification for improving EJB architecture by introducing annotations for EJBs and introducing changes to the EJB Query Language that deals with Entity Bean persistence. EJB 3.0 will make use of the built in annotation functionality in J2SE 5. At the time of writing this JSR was still in draft therefore WebSphere Application Server V6 does not yet support EJB 3.0. It supports all EJB versions up to 2.1 but (in anticipation of EJB 3.0 and the other trends already outlined) it supports EJB annotations through the use of XDoclet technology and its own annotation processing technology.

Conclusion

Expert groups and software vendors concerned with determining that future of the Java Platform promotes and drives annotation-based programming features and functionality. It is, therefore, an area of technology that the developer would do well to invest in to remain current and to enhance productivity. WebSphere Application Server V6 provides tools which anticipate these new industry trends while also supporting previous approaches.

12.3.2 EJB annotations

The WebSphere Application Server V6 environment supports annotations for EJBs. This simplifies the EJB development process which has been somewhat cumbersome. The following sections explain why this is necessary.

EJB advantages: focussing on business logic

Traditionally, when writing back-end applications, the developer was likely to have to implement the following functionality:

- Security
- Database, messaging, e-mail, and other resource management
- Load balancing
- Scalability
- Transactions
- Interfaces to Client Applications

Such considerations distracted the developer from the business logic of the application. Thus, EJB technology in the J2EE platform was developed and taken up enthusiastically by the development community. When implementing back-end logic with EJBs, the developer can largely concentrate on the implementation of the business logic and not the auxiliary functionality listed above.

EJB disadvantages: distraction from business logic

However, as with most things, the benefits of EJBs come at a price. The business logic of an EJB is implemented in a single module (the bean class), but to communicate with the EJB container, and subsequently with wider application server features (such as scalability and transaction management functionality), it is necessary to implement further modules. It is also necessary to configure further parameters in metadata files (or to configure these files via an IDE user interface). These extra artifacts distract the developer from the business logic module itself. Table 12-1 on page 352 gives a list of the interface artifacts required.

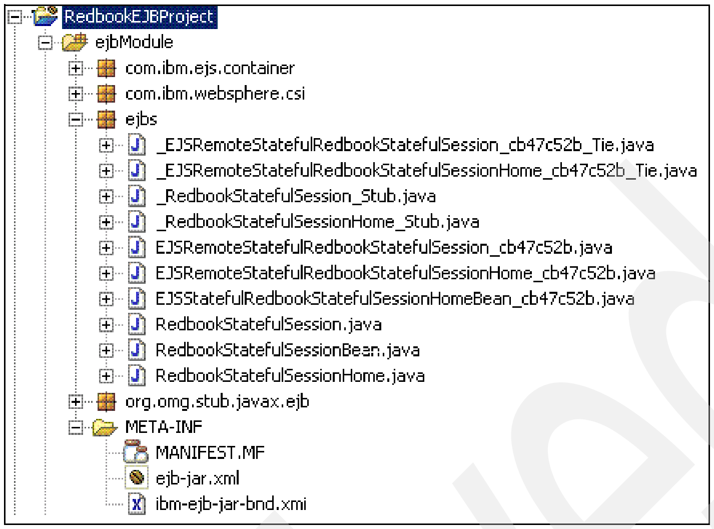
Table 12-1 List of interface types that an EJB (Entity/Session) might have to implement

Interface Type	Example Name	Purpose
home	RedbookHome	Defines EJB life cycle methods
remote	Redbook	Exposes business methods of bean to remote clients
local	RedbookLocal	Exposes business methods for local clients
local home	RedbookLocalHome	Defines EJB life cycle methods for local clients

Note: Not all these interface modules are required in all situations for a particular EJB, but at least two are required. Most enterprise applications require a number of EJBs. Thus, the number of modules to implement soon multiplies.

Further metadata files are required. The ejb-jar.xml is the deployment descriptor for the EJB jar, this Jar file will also contain other files such as its manifest file and also other binding files and in some circumstances files describing the database back end. This swelling number of modules is added to by the Remote Method Invocation (RMI) modules that must also be generated at some point before the EJBs can run in an application server. They allow invocation of EJBs by clients in different JVMs which allows the distribution of enterprise applications.

In an IDE the generation of these artifacts can be handled by Rapid Application Development features such as code generation wizards. Without the use of these IDE features, using a simple text editor program to write all the modules necessary can be at best tiresome and at worst totally impractical. However, even using an IDE, focusing on the business logic module itself remains difficult. Figure 12-1 on page 353 shows the large number of artifacts that are generated even for a single EJB. It is fair to say that not generating deployment code and using a client EJB project can reduce this complexity. Never the less, this example does prove the point.



```
RedbookEJBProject
  ejbModule
    com.ibm.ejs.container
    com.ibm.websphere.csi
    ejbs
      _EJSRemoteStatefulRedbookStatefulSession_cb47c52b_Tie.java
      _EJSRemoteStatefulRedbookStatefulSessionHome_cb47c52b_Tie.java
      _RedbookStatefulSession_Stub.java
      _RedbookStatefulSessionHome_Stub.java
      EJSRemoteStatefulRedbookStatefulSession_cb47c52b.java
      EJSRemoteStatefulRedbookStatefulSessionHome_cb47c52b.java
      EJSStatefulRedbookStatefulSessionHomeBean_cb47c52b.java
      RedbookStatefulSession.java
      RedbookStatefulSessionBean.java
      RedbookStatefulSessionHome.java
    org.omg.stub.javax.ejb
    META-INF
      MANIFEST.MF
      ejb-jar.xml
      ibm-ejb-jar-bnd.xmi
```

Figure 12-1 A single EJB and its metadata, interfaces and generated artifacts

The bottom line is that EJB technology has decreased complexity by standardization but increased complexity by the artifacts required to implement that standardization.

Annotation-based EJB programming

Thus far, the concern of shielding the developer from everything but the business logic has ironically led to a plethora of required modules and a configuration that is defined in more than one file. This situation is where annotation-based programming comes in play. It is a further step in simplifying the enterprise application development process. Annotation-based programming localizes the declaration of bean access and metadata configuration to a single file or artifact that is the Java source file that implements the EJB.

However, in enterprise application development, there is no such thing as a free bean. The use of annotations in EJB development does mean that the developer has to learn a new syntax (annotation doclet syntax) and consider a further alternative (should they use annotations). The WebSphere Application Server V6 environment supports both non-annotation- and annotation-based programming techniques. The motivations for this are to ensure backwards compatibility and choice while at the same time implementing an annotation-based programming approach to increase productivity and ease of deployment. Understanding these motivations is the key to informing the developer on deciding which approach to use. If the developer has existing Enterprise Java programs and existing development experience then it is a reasonable option to ignore annotation-based programming. Previous EJB development techniques are supported. Alternatively, if the developer is new to EJBs, wants to increase productivity, wants easier deployment or simply wishes to explore this new technology then it would be equally reasonable to use this new feature.

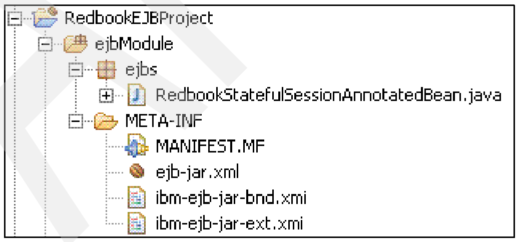
12.3.3 Annotation-based programming

Rational Web Developer, Rational Application Developer, and the Application Server Toolkit support the use of annotation-based programming.

Rational tools

The Rational Web Developer and Rational Application Developer IDEs both support annotation-based programming. When creating an EJB or servlet, the wizard gives the developer the option to support annotations during that object's development. If this option is selected then all metadata for the object is added to its JavaDoc class and method comment blocks. It is also simultaneously generated in the deployment descriptor file. When writing the JavaDoc, code-assist gives the developer the possible tags and their syntax that can be used for the servlet or EJB. The Web or application deployment descriptor file is updated automatically as the developer edits and saves the annotations in the servlet or EJB source file. The developer is oblivious to these updates and can concentrate on their servlet or EJB module as all information pertinent to that module is in front of him in one file.

The IDE further simplifies the developer's view of the EJB by placing any extra modules that are created in a separated project. For example the home, remote and local interface files are placed in a corresponding client project. These interfaces are still in the same package as the bean class itself, but the package is represented in the IDE in both of the projects and only those package members that are pertinent to the project are shown. Not only is all metadata expressed in one file; but also that file is easier to find as it is not shown nested in all the other package files (see Figure 12-2).



```
RedbookEJBProject
  ejbModule
    ejbs
      RedbookStatefulSessionAnnotatedBean.java
    META-INF
      MANIFEST.MF
      ejb-jar.xml
      ibm-ejb-jar-bnd.xmi
      ibm-ejb-jar-ext.xmi
```

Figure 12-2 An annotated EJB with its interfaces in separate project

Supported annotation tags

Annotation tags can be used to annotate Web, EJB, and Web service components and there is a set of defined tags for each of these components types. EJB and Web tags are standard (generic) tags also used by XDoclet. However there are further tags defined which are specific to WebSphere such as WebService tags and tags for specification of container managed relationships. Table 12-2 gives a sample of these tags. This list is not an exhaustive list but rather shows the variety of tags that supported and the naming schemas that are used.

Table 12-2 Sample Tags (not an exhaustive list)

Tag	Component type (IDE-specific or generic)
@WebSphere.WebService	Web Service (WebSphere specific)
@WebSphere.WebMethod	Web Service (WebSphere specific)
@WebSphere.SOAPBinding	Web Service (WebSphere specific)
@WebSphere.EJBBinding	Web Service (WebSphere specific)
@websphere-cmr.join-table	EJB (WebSphere specific)
@websphere-cmr.fk	EJB (WebSphere specific)
@websphere-cmr.column	EJB (WebSphere specific)
@ejb.bean	EJB generic
@ejb.create-method	EJB generic
@ejb.ejb-external-ref	EJB generic
@ejb.ejb-ref	EJB generic
@ejb.env-entry	EJB generic
@web.security-role	Web generic
@web.security-role-ref	Web generic
@web.servlet	Web generic
@web.servlet-init-param	Web generic
@web.servlet-mapping	Web generic
@web.listener	Web generic
@web.filter	Web generic
@web.ejb-ref	Web generic

Sample annotation blocks

Example 12-1 illustrates a sample EJB annotation block. The annotation is in the class JavaDoc block. There are separated tags for the bean and for its home and remote interfaces.

Example 12-1 EJB annotation block

```
/**
 * Bean implementation class for Session Bean: RedbookStatefulSessionAnnotated
 *
 * @ejb.bean
 *  name="RedbookStatefulSessionAnnotated"
 *  type="Stateful"
 *  jndi-name="ejb/ejbs/RedbookStatefulSessionAnnotatedHome"
 *  view-type="remote"
 *  transaction-type="Container"
 *
 * @ejb.home
 *  remote-class="ejbs.RedbookStatefulSessionAnnotatedHome"
 *
 * @ejb.interface
 *  remote-class="ejbs.RedbookStatefulSessionAnnotated"
 *
 */
```

Example 12-2 illustrates a sample servlet annotation block. In this sample, the class javadoc block has entries for the servlet, for the servlet mapping, and an initialization parameter for the servlet.

Example 12-2 Servlet annotation block

```
/**
 * Servlet implementation class for Servlet: RedbookServlet
 *
 * @web.servlet
 *   name="RedbookServlet"
 *   display-name="RedbookServlet"
 *
 * @web.servlet-mapping
 *   url-pattern="/RedbookServlet"
 *
 * @web.servlet-init-param
 *    name="Redbook"
 *    value="IBM WebSphere V6 Planning and Design"
 *    description="The title of the Redbook in question."
 *
 */
```

12.4 Deployment automation

The second key part of WebSphere Application Server V6 rapid application deployment is deployment automation. Deployment automation allows the automatic installation of applications and modules to run local or remote application server or servers.

12.4.1 Deployment model features

The following are some of the features of the new automated deployment model:

- Headless and hot-directory

  Currently WebSphere Application Server V6 environment supports a headless, hot-directory model. *Headless* means that there is no GUI for this deployment facility. *Hot-directory* means that the files to be deployed are saved to a specific directory that is monitored by the deployment process. When files are updated in this directory, they are installed automatically to the application server.

- Automated EAR management

  When an application is added or updated in the hot-directory, the deployment process makes default decisions for all unspecified deployment settings. These decisions minimize the complexity that a developer or administrator deals with when deploying an application. This concept is known as *automated EAR management*.

- By-part application construction

  The application does not have to be packaged in an Enterprise Java archive such as an EAR or WAR file for the deployment to work. Artifacts for a particular application can be dropped into the hot-directory without any Enterprise Java packaging and still be deployed. Any required artifacts are generated from the existing artifacts. If only a small part of the application is updated then only that part of the application is changed on the server. The deployment process updates the application server by application update and JMX commands.

  This approach means that fine-grained and incremental changes can be made to the application without the developer having to concern himself with repackaging the application each time. This also means the entire code of an application is not affected when a small part of the application is updated. This concept is known as *minimal application impact.*

12.4.2 Styles of deployment

Because of the auto-management of archives and the by-part construction of applications, there are two styles for deployment that can be specified when setting up automated deployment.

- Automated auto-app install

  This style is where the developer provides enterprise archive packages. That is the developer provides an EAR and or WARs, EJB Jars, Client Jars. When these files are placed in the deployment directory, the WebSphere Rapid Deployment process then deploys these components to the corresponding application server. Deployment options in the provided deployment descriptors, where present, are used. When not present, the automated EAR management feature generates default values.

- Free-form project install

  This style is where the developer provides unpackaged artifacts such as JSPs, HTML, servlets, and EJB source code. The WebSphere Rapid Deployment process in this case generates the J2EE artifacts and packages which have not been provided. It is possible for settings to be specified in annotations in the Servlets and EJB code for example, in this case these annotations are processed when generating the deployment descriptors and packaging. This style of install is best combined with annotation-based programing because it gives the developer more ease of installation while maintaining control over configuration.

12.4.3 Setting up deployment automation

Setting up deployment automation is done under the WebSphere Application Server. It is a headless process which means there is no GUI. Instead scripts are used. There are two steps to the set up.

- Configure a WebSphere Rapid Deployment project using the wrd-config script. You must set up an environment variable called WORKSPACE to point to the directory where projects are to be stored.

  The administrator or developer must specify the eclipse project name to be installed and the deployment style for this project (automated auto-app install or free-form project install). An interactive command line interface then prompts the user for additional information:

  – The name of the server to deploy to.

  – The JMX host name which the host containing the server to deploy to (or in a network deployment environment the host name of the Deployment Manager).

  - The JMX SOAP port that it listens on (or in a network deployment environment this should be the JMX SOAP port of the Deployment Manager).
  - If global security is turned on, then the security user ID and password is also required.
- Run the WebSphere Rapid Deployment process itself. This process can be executed in a batch or monitor mode. In a batch mode the process will run a single build of the Rapid Deployment work space and then shuts down, no deployment of the application is attempted. In monitor mode the process runs indefinitely, monitoring the work space directory for updates. When new updates are detected, the project is rebuilt and then deployed to the application server specified in the configuration step.

Attention: If the -monitor option for rapid deployment is selected, then the process is long running. If an application is removed from the monitored directory, then the rapid deployment daemon uninstalls the application from the target server. This is a daemon that should not be running in a production environment to avoid the inadvertent uninstall of sensitive applications.

Note: You can use WebSphere Rapid Deployment to deploy applications to stand-alone application servers (that is those servers that are not part of a cell). You can also use it to deploy to servers that are part of a cell (that is, WebSphere Application Server Network Deployment V6 servers). However, the wrd application does not allow you to deploy applications to a server that is a member of a cluster. WebSphere Rapid Deployment is currently aimed for use in a development environment and possibly a test environment rather than in production. (Actually, even test environments are likely to mirror production environments, which means they are likely to have clusters.)

12.5 Best practices for deployment

Application deployment is comprised of two basic steps:

- Configuring the application server or servers with required resources for the application.
- Installing the application to the application server or servers.

These steps can be achieved in a number of ways. This section gives an overview of all the deployment alternatives and when and where they might be best employed.

The following methods are available for deployment:

- Rapid application deployment
- Deployment via Rational Application Developer
- Deployment via the WebSphere administrative console
- Deployment using **`wsadmin`**
- Deployment using **`Ant`**

Note: These methods are essentially either command line or GUI interfaces that use JMX and MBeans to interact with the WebSphere Application Server environment.

12.5.1 Rapid application deployment

The main benefit of rapid application deployment is that it shields the administrator from the complexity of the WebSphere administrative console, **`Ant`**, and **`wsadmin`**. However, like **`Ant`** and **`wsadmin`**, there is no GUI for this deployment option.

The auto-app option

The auto-app option allows the developer or administrator to place enterprise packaged applications into a directory under the WebSphere Application Server. The applications are then installed or updated on the application server. This approach is best used when the developer or administrator is comfortable with Java packaging and is more appropriate for a test environment. You should note, however, that removing the application archives from the hot directory uninstalls the application from the application server. Using the monitor mode for the rapid deployment demon is not best practice in a production environment as it might lead to unintended uninstall of applications.

The free-form option

The free-form option allows the developer to deploy code without concerning himself with Enterprise packaging. It also allows for fine grained, low application impact updates. This approach is perhaps best suited (but not exclusive) to the development environment where the developer needs to have fine grained control over what artifacts are being deployed and does not want to concern themselves with enterprise packaging. In such an environment, the monitor option for the rapid application development demon is a good option as code is likely to be updated regularly to the monitored directory. This approach is also well suited to use in combination with annotation-based programming which is another reason why it fits with the development environment.

Advantages

The following are the advantages to rapid application deployment:

- There are less configuration complexity and choices.
- The autoapp option might be better for test environment deployment.
- The free-form option is useful during development.

Disadvantages

The following are the disadvantages to rapid application deployment:

- There is no GUI. Therefore, rapid application deployment is less intuitive.
- A process that makes choices for you is less under your control.
- Applications can be inadvertently uninstalled from the application server if the install package is deleted from the hot directory.
- This method does not deploy to clustered application servers.

12.5.2 Deployment via Rational Application Developer

Rational Application Developer has a Server Tools feature which is a part of the Application Server Toolkit, a subset of the Rational Application Developer functionality. The Server Tools feature allows you to publish applications in Rational Application Developer projects to separate local and remote WebSphere Application Servers. This deployment method is best suited to a development or test environment.

To set up an environment, you first have to define the external servers in Rational Application Developer. Then, you can start and stop the local or remote servers and publish projects to the servers.

You can run, debug, or profile the project on the remote server. Run means simply running the application as it would be run in a production environment to check functionality. Debugging allows analysis of code execution flow to find the cause of errors in execution. Profiling means to run the application to collect performance data on the application. To profile the application on a remote server, an extra application called an Agent Controller must be installed and running on the remote WebSphere Application Server machine. The Agent Controller install image comes with Rational Application Developer.

Tip: If the Agent Controller is required then it should be installed first before installing the remote server on the remote machine. Other wise, the configuration for the Agent Controller has to be updated manually to tell it about the application server.

Advantages of deployment via Rational Application Developer

With Rational Application Developer, you can:

- Deploy applications to local or remote application servers directly from the IDE.
- Run applications as you would in production.
- Run applications in debug mode.
- Profile applications on the server to gather performance data.

Disadvantages of deployment via Rational Application Developer

The disadvantages of deployment via Rational Application Developer are:

- Configuration of resources on the application server has to be done via the administrative console of the server not via the Rational Application Developer interface.
- For profiling the agent controller, the application must be installed on remote server machines, which is an extra overhead.

12.5.3 Deployment via the WebSphere administrative console

WebSphere Application Server provides a GUI application called the administrative console for administering the WebSphere Application Server environment.

WebSphere Application Server V3.5 provided a Web-based console. Version 4.0 provided a GUI based on Java Swing application. With the V5, the Web-based GUI was reverted to and significantly enhanced. WebSphere Application Server V6 continues this enhancement. The reason for the console staying Web-based is that it allows for remote administration with a thin-client rather than deploying a client application to the end user's machine or using a remote application windowing application.

The administrative console is a J2EE application itself and is installed with the application server. The administrative console for WebSphere Application Server V6 can be reached at: http://<hostname>:9060/admin.

Note: In WebSphere Application Server V6, the initial port for the administrative console has changed to 9060.

The Network Deployment environment

In a Network Deployment environment, a cell is created and application servers are added to that cell. The purpose of a WebSphere Application Server cell is for all entities in the cell to be administered from a single point — the administrative interfaces of the Deployment Manager. Therefore, each cell has only one Deployment Manager.

To deploy an application there are two basic steps:

- Configure resources for the application

  You need to set up any resources that an application might use under the Resources menu. Examples of resources are:

  – Database providers and data sources
  – Message servers, queues, and listeners
  – E-mail resource providers

- Install the application

  Then, you install applications via the WebSphere administrative console, which runs on the Deployment Manager node, from the Applications menu.

  The install interface allows you to upload the EAR or WAR application archive from:

  – The file system that is local to the Web browser (local file system).
  – A remote file system which might be any node in the cell.

Note: It is best practice to manage your applications archives from a single file system such as on the Deployment Manager machine or a development machine. However, the remote file system option does mean that files can be uploaded from the file system of any node in the cell.

A series of steps are gone through when the archive has been uploaded. These steps allow you to map resources in the application server to components in the application. For example, data sources are mapped to entity beans. Changes are then saved, and at this point, the changes can be synchronized across all nodes that are members of the cell.

Stand-alone server

Stand-alone servers are not managed by a Deployment Manager. They have their own administrative application installed. Therefore, when administering a stand-alone server, you should use the administrative console accessing directly to that server. The advantage of the stand-alone server is that it allows the deployment of an application to a server or servers via a thin client that can be accessed by any authorized user with a Web browser and network connectivity. The disadvantage is that repetitive or more complex deployments have many configuration steps that need to be remembered and repeated. In these cases, you should consider command line scripting installation options.

12.5.4 Deployment using wsadmin command line scripting

It is possible to run scripts on the command line to configure WebSphere Application Server. This option can present a useful alternative to using the administrative console. The deployment procedures of resource configuration and then application installation can be executed in a single command that runs a script.

The wsadmin command

Commands are run via the **`wsadmin`** executable that is found in the bin directory under WebSphere Application Server installation directory. The **`wsadmin`** executable can be:

- Run as a command line prompt where you enter commands.
- Invoked with a script as a parameter which it then executes and exits.

Command Language (Tcl and Jacl)

The syntax of the commands and scripts submitted to **`wsadmin`** is Tool Command Language (Tcl). Tcl is an interpreted scripting language. Java Command Language (Jacl) is simply a Java implementation of Tcl. The **`wsadmin`** command uses the Jacl implementation.

Tip: When looking for guides to syntax for Jacl, look for Tcl syntax guides. Remember, Jacl is a Java implementation of Tcl.

Scripts that are written in this syntax can define procedures and variables, control flow structures and all the other structures one would associate with a scripting language. You can find more information about Tcl and its Java implementation at:

`http://www.tcl.tk/software/java`

The wsadmin objects

While Tcl has a generic programming syntax, **`wsadmin`** has defined Tcl objects that are specific to WebSphere. These objects are:

- Help
- AdminApp
- AdminConfig
- AdminControl

These are the objects that you manipulate to configure your server.

Tip: The first of these objects is the Help object. Therefore, when starting out with **`wsadmin`** scripting, run **`wsadmin`** at the command line and then enter `$Help`. This option gives more details about what objects and parameters are specific to WebSphere. They are also documented in the information center at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

The Jacl script shown in Example 12-3 on page 366 illustrates what is possible with **`wsadmin`** and Jacl. The script:

- Checks arguments for node, server and WAR file location.
- Prints usage if arguments are not specified and exits.
- Specifies the deployment descriptor location in the WAR file.
- Sets up context root and virtual host mapping for the Web application in the WAR file that is specified.
- Installs the application on the specified node and server.
- Saves this updated configuration.

You would run this script from the command line using the following syntax:

```
wsadmin.bat/sh <the jacl script location> <node> <server> <war>
```

Note: The node, server, and war arguments are for the Jacl script, not **`wsadmin`**. The Jacl script location is the argument for **`wsadmin`**.

Example 12-3 Jacl Script for installing an application, with command line options

```
# Java Command Language Jacl script for installing Redbook web application
# set up some variables
   set usage “wsadmin.bat -f <Jacl file path> <node> <server> <warFile>“
   set node ““
   set server ““
   set warFile ““
# Define a procedure for checking arguments
proc checkArguments {} {
   global argv[]
   global node
   global server
   global warFile

   set node [lindex $argv 0]
   set server [lindex $argv 1]
   set warFile [lindex $argv 2]

   if{$node == ““} {
      puts “Missing argument: node”
      puts “usage“
      return 1
   }
   if{$server == ““} {
      puts “Missing argument: server“
      puts “usage“
      return 1
   }
   if{$warFile == ““} {
      puts “Missing argument: warFile“
      puts “usage“
      return 1
   }
   return 0
}
# read in arguments and check they are ok
set argcheck[checkArguments]
if {$argcheck ==1 }{ exit}
# install application in the war file on the specified server and node
set modtovh1 [list “RedBookWebModule“ $warFile,WEB-INF/web.xml default_host]
set modtovh [list $modtovh1]
set attrs [list -contextroot RedBook -appname RedBookApp -MapWebModToVH
$modtovh -node $node -server $server]
$AdminApp install RedBook.ear $attrs
$AdminConfig save
```

The script in Example 12-3 on page 366 is informative to the deployment discussion in three ways:

- The calls to the AdminConfig and AdminApp objects at the last two lines of the script show the usage of these WebSphere-specific objects. In this case, they install the application to the server and save the server configuration.
- A node and server are specified. Scripting can be used in a Network Deployment environment very flexibly to deploy applications on specific nodes and servers.
- The procedure (checkArguments{}) has been defined to check the input arguments for this script. This procedure is intended to illustrate that Jacl/Tcl is a fully-fledged scripting language that can be used to create sophisticated scripts that can express what ever the developer or administrator requires.

Advantages of Command Language

The advantages of using Command Language are:

- It is useful for running repetitive and complex tasks.
- It is implemented using Tcl which is a widely used scripting language.
- Commands can be run individually or as a script.
- After you learn this scripting tool, it is very powerful.

Disadvantages of Command Language

The disadvantages of using Command Language are:

- There is no GUI so it is less intuitive.
- Tcl can initially seem complex.
- You have to learn the specific WebSphere object syntax.

12.5.5 Deployment using Ant

`Ant` is an open source Java based build tool from Apache. It is platform independent as it uses Java and XML technology.

Java technology is used to write **`Ant`** tasks. **`Ant`** comes with many ready-made tasks such as CVS interaction, file I/O tasks, and Java compilation tasks. **`Ant`** comes with WebSphere Application Server, which also provides specific Java tasks for WebSphere administration. For deployment, for example, the key WebSphere **`Ant`** task is InstallApplication, which installs applications to a cell or an application server. For a full list of WebSphere specific **`Ant`** tasks, see:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

To run **Ant** tasks, you invokes the **was_ant** executable in the bin directory of the application server and give this program the name of an XML build file. This build file has target blocks of XML. Each block specifies jobs to be done in the build process. Each block makes use of the available **Ant** and WebSphere tasks.

Ant was originally intended for use as a build tool. However, because it can have custom tasks created for it (such as the WebSphere **Ant** tasks), it was soon used for more than just building applications. Therefore, the name build file for the XML file is misleading. The build file might not do any application building at all. It might, for example, take an EAR file and install it on an application server as illustrated in Example 12-4.

Example 12-4 Ant build XML file with target blocks to install or uninstall an application.

```
<!-- Ant build file for installing/uninstalling the Redbook application -->

<!-- install the redbook application with this target block -->

<target name="installRedbookApp" >
<taskdef name="wsInstallApp"
classname="com.ibm.websphere.ant.tasks.InstallApplication"
classpath="C:\WebSphere\AppServer\lib\wsanttask.jar"/>
<wsInstallApp wasHome="C:/WebSphere/AppServer"
ear="C:/temp/install/Redbook.ear">
</wsInstallApp>
<echo>Completed Redbook install</echo>
</target>

<!-- uninstall the redbook application with this target block -->

<target name="uninstallRedbookApp">
<taskdef name="wsUninstallApp"
classname="com.ibm.websphere.ant.tasks.UninstallApplication"
classpath="C:\WebSphere\AppServer\lib\wsanttask.jar"/>
<wsUninstallApp wasHome="C:\WebSphere\AppServer" application="Redbook">
</wsUninstallApp>
<echo>Completed Redbook uninstall</echo>
</target>
```

Note: This example can be invoked by running the WebSphere **Ant** executable which is in the bin directory under all application servers. The **Ant** executable takes the file in this example as its build file parameter. It also takes the name of each target block as a parameter. The targets are then run in that order. If the target block name is not given as a parameter, the block is not run.

Generally, when WebSphere developers and administrators talk about **`Ant`** and **`wsadmin`**, **`Ant`** is usually associated with building applications, and **`wsadmin`** is associated with WebSphere Application Server configuration and management, including installing applications. However, there is also a WebSphere **`Ant`** task that allows invocation of **`wsadmin`**. Therefore, the combination of WebSphere **`Ant`** and **`wsadmin`** allows the developer and the administrator to create flexible and extensive script libraries to automate tasks.

For further details about the Apache **`Ant`** Project, see:

`http://ant.apache.org`

Advantages

The advantages of deployment using **`Ant`** are:

- **`Ant`** is open source and widely used for many purposes.
- Users can create their own Java **`Ant`** tasks if need be.
- **`Ant`** scripts are useful for complex or repetitive tasks.
- **`Ant`** can build as well as deploy applications potentially in one script.
- **`Ant`** can invoke **`wsadmin`**, a flexible and powerful combination.

Disadvantages

The disadvantages of deployment using **`Ant`** are:

- You must learn the **`Ant`** build file XML syntax in order to write build scripts, another overhead for the developer or administrator.
- **`Ant`** and wsadmin script combinations can be complicated, requiring knowledge of both syntaxes.
- There is no GUI, so this method is less intuitive.
- Some developers prefer to use **`Ant`** purely for building applications and **`wsadmin`** Jacl for deployment to keep these scripts and processes separate.

13

Planning for system management

This chapter provides an overview of the planning necessary for the system management of the WebSphere Application Server V6 runtime environment. It focuses on developing a coherent strategy to best use the multitude of system management capabilities in WebSphere Application Server V6. The operational efficiency of the overall system hinges upon the proper implementation of the system management processes. This chapter contains the following sections:

- System administration overview
- Configuration planning
- Planning for application administration
- Plan for backup and recovery
- Plan for availability
- Plan for performance
- Plan for security
- Summary

13.1 System administration overview

WebSphere Application Server V6 provides a variety of administrative tools for deploying and administering your applications and application serving environment, including configurations and logical administrative domains. WebSphere Application Server nodes and cells can be administered using a combination of the following tools:

- Administrative console
- WebSphere scripting client (**`wsadmin`**)
- Using **`Ant`** to automate tasks
- Using administrative programs
- WebSphere Application Server command line tools

Note: All the administrative options that are available are dependent on the WebSphere profiles that are configured. For further details, refer to 3.2.3, “WebSphere profiles” on page 52.

13.1.1 Administrative console

The WebSphere Application Server administrative console connects to a running single server or to a Deployment Manager and is accessed through a Web browser via the following URL:

```
http://<hostname>:9060/ibm/console
```

If WebSphere Application Server security is enabled, you must then enter your user name and password to access the administrative console. In this case, the actions that you are able to perform within the administrative console are determined by your role assignment. The following roles apply to access control within the administrative console:

- Monitor

  The Monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

- Configurator

  The Configurator role has the same permissions as the Monitor and can change the WebSphere Application Server configuration.

- Operator

  The Operator role has same permissions as the Monitor and can start or stop services.

- Administrator

  The Administrator role has the combined permissions of the Operator and Configurator.

If WebSphere Application Server security is not enabled, you must still enter a user name to gain access to the administrative console. This user name is not validated and is used exclusively for logging purposes and to enable the system to recover the session if it is lost while performing administrative tasks.

Each console user has a user workspace. That user workspace is stored in <install_root>/wstemp/<USER>/workspace.

13.1.2 WebSphere scripting client (wsadmin)

The WebSphere administrative (**`wsadmin`**) scripting program is a powerful, text console-based command interpreter environment that enables you to run scripted administrative operations. The supported scripting languages are Jacl (a Java implementation of Tcl) and Jython (a Java implementation of Python). The **`wsadmin`** tool can be run in interactive and unattended mode. You can use the **`wsadmin`** tool to perform the same tasks that you can perform using the administrative console.

The **`wsadmin`** tool gives access to several scripting objects, allowing administration of various WebSphere Application Server areas. These objects can be used for application management, configuration, operational control, and for communication with MBeans that run in WebSphere Application Server processes.

13.1.3 Using Ant to automate tasks

WebSphere Application Server V6 provides a copy of the **`Ant`** tool and a set of **`Ant`** tasks that extend the capabilities of **`Ant`** to include product-specific functions. **`Ant`** has become a very popular tool among Java programmers.

Apache **`Ant`** is a platform-independent Java-based build automation tool, configurable through XML script files and extensible through the use of a Java API. In addition to the base **`Ant`** program and tasks, WebSphere Application Server V6 contains a number of tasks that are specific to WebSphere Application Server to manage and build applications.

The **Ant** environment within WebSphere Application Server V6 allows you to create platform-independent scripts that compile, package, install, and test your application on the application server. For example, the available tasks that can:

- Install and uninstall applications.
- Start and stop servers in a base configuration.
- Run administrative scripts or commands.
- Run the Enterprise JavaBeans (EJB) deployment tool.
- Run the JavaServer Pages (JSP) file precompilation tool.

13.1.4 Using administrative programs

WebSphere Application Server supports access to the administrative functions through a set of Java classes and methods. You can write a Java program that performs any of the administrative features of the WebSphere Application Server administrative tools. You can also extend the basic WebSphere Application Server administrative system to include your own managed resources.

You can prepare, install, uninstall, edit, and update applications through programming. Preparing an application for installation involves collecting various types of WebSphere Application Server specific binding information to resolve references that are defined in the application deployment descriptors. This information can also be modified after installation by editing a deployed application. Updating consists of adding, removing or replacing a single file or a single module in an installed application, or supplying a partial application that manipulates an arbitrary set of files and modules in the deployed application. Updating the entire application uninstalls the old application and installs the new one. Uninstalling an application removes it entirely from the WebSphere Application Server configuration.

13.1.5 WebSphere Application Server command line tools

There are several command line tools that you can use to start, stop, and monitor WebSphere Application Server server processes and nodes. These tools only work on local servers and nodes. They cannot operate on a remote server or node. To administer a remote server, you can use the **wsadmin** scripting program, as described in 13.1.2, “WebSphere scripting client (wsadmin)” on page 373, connected to the Deployment Manager for the cell in which the target server or node is configured.

All command line tools function relative to a particular profile. If you run a command from the directory <install_root>/WebSphere/AppServer/bin, the command will run within the default profile.

13.2 Configuration planning

One of the key elements to a successful WebSphere Application Server V6 implementation is proper configuration planning. There are many aspects of the WebSphere Application Server V6 runtime environment that needs to be considered. Be sure to understand how the changes will impact the environment before making any configuration changes. The basic administrative architecture should be mastered in order to plan configuration changes.

13.2.1 Administrative architecture

The fundamental concepts for WebSphere Application Server administration include:

- Software processes called *servers*.

  Servers perform the actual running of the code. Several types of servers exist depending on the configuration. Each server runs in its own JVM. The application server is the primary runtime component in all WebSphere Application Server configurations.

- Topological units referenced as *nodes* and *cells*.

  A node is a logical group of WebSphere Application Server managed server processes that share a common configuration repository. A node is associated with a single WebSphere Application Server profile. One machine can host one or more nodes, but a node cannot span multiple machines. A node can contain zero or more application servers.

- The *configuration repository* used for storing configuration information.

  The configuration repository holds copies of the individual component configuration documents that define the configuration of a WebSphere Application Server environment. All configuration information is stored in XML files.

13.2.2 Configuration repository

A configuration repository stores configuration data. A cell level repository stores configuration data for the entire cell and is managed by a file repository service that runs in the Deployment Manager. The Deployment Manager and each node have their own repositories. A node-level repository stores configuration data needed by processes on that node and is accessed by the node agent and application servers on that node.

When you change the cell configuration from the Deployment Manager, for example by creating an application server, installing an application, changing a variable definition or the like, and then save the changes, the cell level repository is updated. The file synchronization service will distribute the changes to the appropriate nodes.

The directory in which a configuration file exists determines the scope of the data in the configuration file. Files in an individual server directory apply to that specific server only. Files in a node level directory apply to every server on that node. Files in the cell directory apply to every server on every node within the entire cell. The files are replicated to every node in the cell. Several different configuration settings apply to the entire cell including the definition of virtual hosts, shared libraries, and any variables that you want consistent throughout the entire cell.

Distributed process discovery

One of the new features of WebSphere Application Server V6 is the ability to manage the startup of applications and servers. Two new settings enable you to fine-tune the startup speed and order of applications that are configured to start automatically when the server starts:

- The *starting weight* setting lets you specify the order in which to start applications when the server starts. This is not a new parameter, but in earlier versions of WebSphere Application Server, this parameter was not available through the administrative console.
- The *background application* setting lets you specify whether an application must initialize fully before its server starts or if the server can proceed without waiting for the application.

The Deployment Manager, Node Agents, and application servers can start in any order they are discovered with the exception that the node agent must start before any application server on that node. Communication channels are established as they startup and each has its own configuration and application data to start. Performance can be vastly increased with proper management of the startup process.

Simplified replication process

There is a new type of replication domain that is based on the new high availability framework. By using data replication domains, you do not have to configure local, remote, and alternate replicators. Any replication domains that were created with a previous version of WebSphere Application Server might be multi-broker domains. Existing multi-broker domains remain functional, however, after you upgrade your Deployment Manager, you can create only data replication domains in the administrative console. For improved performance and easier configuration, migrate any existing multi-broker domains to the new data replication domains.

Common networking service

The new channel framework model provides a common networking service for all components, including IBM service integration technologies, WebSphere Secure Caching Proxy and the high availability manager core group bridge service. This model consists of network protocol stacks or transport chains that are used for communication within an application server environment.

Transport chains consists of one or more types of channels, each of which supports a different type of communication protocol, such as Transport Control Program (TCP), Distribution and Consistency Services (DCS), or HyperText Transport Protocol (HTTP). Network ports can be shared among all of the channels within a chain. The channel framework function automatically distributes a request arriving on that port to the correct communication channel for processing.

Server configuration templates

New in WebSphere Application Server V6 is the ability to create a customized server template that is based on an existing server configuration. Server templates can then be used to create new servers. This provides a powerful mechanism to propagate the server configuration both within the same cell, as well as across cell boundaries. In order to propagate the server configuration across cell a boundary template, the server configuration must be exported to a configuration archive, after which it can be imported to another cell.

Resource scopes

The scope of a resource can be altered in the administrative console according to needs but must adhere to a certain set of rules:

- The application scope has precedence over all the scopes.
- The server scope has precedence over the node, cell, and cluster scopes.
- The cluster scope has precedence over the node and cell scopes.
- The node scope has precedence over the cell scope.

The scope can be changed to specify the level at which a resource is visible with the administrative console panel. A resource can be visible in the collection table at the cell, node, cluster, or server scope. By changing the scope, you can see other variables that apply to a resource and might change the contents of the collection table. Resources are always created at the currently selected scope, even though the resources might be visible at more than one scope.

Resources, such as JDBC providers, namespace bindings, or shared libraries, can be defined at multiple scopes. Resources that are defined at more specific scopes override duplicate resources that are defined at more general scopes. In other words, resources with a smaller scope takes precedence over resources with a larger scope. Despite the scope of a defined resource, the resource properties only apply at an individual server level. If you define the scope of a data source at the cell level, all the users in that cell can look up and use that data source. However, resource limits defined for cell level resources are limits for each server in the cell. For example, if you define the maximum number of connections to be 10, then each server in that cell can have 10 connections.

You can configure resources and WebSphere Application Server variables under all five scopes. You can configure namespace bindings and shared libraries only under cell, node, and server scopes.

The following are the scope levels, listed in order of granularity with the most general scope first:

- Cell scope

  The cell scope is the most general scope and does not override any other scope. It is recommended that cell scope resource definitions should be further granularized at a more specific scope level. When you define a resource at a more specific scope, you provide greater isolation for the resource. When you define a resource at a more general scope, you provide less isolation. Greater exposure to cross-application conflicts occur for a resource that you define at a more general scope.

  The cell scope value limits the visibility of all servers to the named cell. The resource factories within the cell scope are defined for all servers within this cell and are overridden by any resource factories that are defined within application, server, cluster and node scopes that are in this cell and have the same Java Naming and Directory Interface (JNDI) name. The resource providers that are required by the resource factories must be installed on every node within the cell before applications can bind or use them.

- Cluster scope

  The cluster scope value limits the visibility to all the servers on the named cluster. All cluster members must be running WebSphere Application Server V6 to use cluster scope for the cluster. The resource factories that are defined within the cluster scope are available for all the members of this cluster to use and override any resource factories that have the same JNDI name that is defined within the cell scope. The resource factories that are defined within the cell scope are available for this cluster to use, in addition to the resource factories, that are defined within this cluster scope.

- Node scope (default)

  The node scope value limits the visibility to all the servers on the named node. This is the default scope for most resource types. The resource factories that are defined within the node scope are available for servers on this node to use and override any resource factories that have the same JNDI name defined within the cell scope. The resource factories that are defined within the cell scope are available for servers on this node to use, in addition to the resource factories that are defined within this node scope.

- Server scope

  The server scope value will limit the visibility to the named server. This is the most specific scope for defining resources. The resource factories that are defined within the server scope are available for applications that are deployed on this server and override any resource factories that have the same JNDI name defined within the node and cell scopes. The resource factories that are defined within the node and cell scopes are available for this server to use, in addition to the resource factories that are defined within this server scope.

- Application scope

  The Application scope value limits the visibility to the named application. Application scope resources cannot be configured from the administrative console. Use the Application Server Toolkit or the **wsadmin** tool to view or modify the application scope resource configuration. The resource factories that are defined within the application scope are available for this application to use only. The application scope overrides all other scopes.

13.2.3 Application life cycle and naming conventions

Naming conventions make the runtime environment more comprehensible by making them easier to understand. They can also provide information about the particular component or function. Using a consistent naming convention helps standardize the structure of the environment and allow for easy expansion.

Application life cycle model

An industrial strength WebSphere Application Server site topology is not limited to a production system, but rather a set of systems for development, testing, and operation of J2EE applications. The application life cycle model for these applications is closely linked to the processes, tools, and technologies used during application development and the requirements for testing. On the other hand, the life cycle model cannot be discussed independently of the size and topology of the future production site, which strongly depends on the non-functional requirements.

Tip: The more complex the production topology, the more stages that are required for full testing of the applications.

A small WebSphere Application Server installation might cover the requirements for a single project, single application, single version development model. A large WebSphere Application Server installation might have to cope with a multi-project, multi-site, multi-supplier, multiple subsystem integration, multi-version, multi-release development model and lots of related issues. Where a small installation could even fit onto a single computer, a large installation typically involves multiple machines and even geographically distributed locations.

Review the number of parallel projects and the number of simultaneously available releases, and then decide on your staging model. The topology topology concepts of cell, node, server, and cluster, available within the WebSphere Application Server infrastructure help structure the requirements that are derived from the application life cycle, development model, and technologies that are used.

Naming concepts

Developing, establishing, and maintaining good naming concepts for all of the setup and configuration information of a WebSphere Application Server site is an important part of the installation planning.

All parts of the installation planning, particularly the naming concepts, should be discussed with the subject matter experts that are involved, and decisions and results should be documented in writing. Failure to do so can affect operational processes for the future site, and has proved in the past to be a root source for labour-intensive issues within customer installations.

The following sections discuss some considerations that you should take into account when developing the key concepts for the site during installation planning.

Naming for WebSphere Application Server

Consider including the following parameters as part of your WebSphere Application Server naming concept:

- Cell, WebSphere Application Server cell
- System, server or logical partition where the cell or node is located
- Node, WebSphere Application Server node
- Application life cycle stage
- Type

Naming for applications

Mandatory naming conventions for all parts of future J2EE applications that are exposed to system management in one way or another can help staff that is involved to perform in a more efficient way.

As a preparatory step, the concepts roles of J2EE application assembler and application deployer must be studied and fully understood. A key requirement for application naming is good navigation using filters in the admin application view and scripting API. If the number of applications is going to be large, consider using some extra grouping element as part of the naming concept or to use separate nodes for the different application life cycle stages.

The application naming concept should include stage name and some naming key which will link all application configuration items. Because the Enterprise Archive (EAR) file names of the deployed applications limitations have to be unique within a cell, consider using the application life cycle stage as a global prefix. A short form of the application name should link all configuration elements. A subsystem identifier is good for determining the origin or destination of JNDI namespace bindings. Decide on the scope of your bindings early on, as this will be difficult to clean up later on. Refer to Table 13-1 for an example of a naming scheme for deployable assets.

Table 13-1 Example naming schemes for deployable assets

Archive type	Naming template
Enterprise Archive (EAR)	<stage>_<appl>_<appl_long_name>_EAR.ear
Web Application Archive (WAR)	<appl>_<war_long_name>_WAR.war
Enterprise Java Bean (EJB)	<appl>_<ejb_long_name>_EJB.jar
Java Archive (JAR)	<appl>_<jar_long_name>_JAR.jar

For a better understanding of your requirements, consider the point between visibility of information related to your application assembly and its ingredients. For the elements contained within, it is desirable to keep them invariant under stage change, unless you plan for fully automated build, deployment and configuration processes to handle the increased complexity.

For example, if you decide to keep your data sources and EJB references at the server scope, use jdbc/W<appl>_<datasourcename> and ejb/W<appl>_<ejbname>, where <appl>, <datasourcename>, and <ejbname> allow to link back to the deployed EAR file by using a default mapping of declared references to configured resources.

Naming for security

Setting up a WebSphere Application Server infrastructure involves considering several aspects of technical security, in particular on a UNIX or zOS System:

- Basic network security

  On zOS, you need to know your address space names and ports that each instance uses to allow port binds.

- WebSphere Global Security

  Ensure WebSphere Application Server installation will not be modified unduly.

- Separation of concern

  Where necessary, use SSL with different certificates to establish different security and trust zones for namespace publishing, lookups, and so forth.

This ties in with the WebSphere Application Server naming concept. Establish a consistent mapping between application server name, process name, process owner, user ID, group ID, and port range to be used. Key ring and certficate names should also be derived from this naming concept.

Refer to *WebSphere Application Server V6 - Security Handbook*, SG24-6316 for a more in-depth discussion of application security.

Port assignment scheme

The port assignment scheme for a WebSphere Application Server site should be developed in close cooperation with the network administrators. From a security point-of-view, it is highly desirable to know each and every port usage beforehand, including the process names and owners using them. Depending on the chosen WebSphere Application Server and hardware topology, the setup even for a single machine could involve having multiple cells, nodes, and server instances in a single process space. Each WebSphere Application Server process requires exclusive usage of several ports, and knowledge of certain other ports for communication with other processes.

13.2.4 Change management

Proper change management is important to the longevity of any application environment. WebSphere Application Server V6 contains a number of technologies to aid with the change management process. WebSphere Application Server V6 permits more fine-grained updates to applications. It allows application components to be supplied and the restart of only parts of the application. This preserves application configuration during the update process.

Application updates

With WebSphere Application Server V6, you can update application files deployed on a server. Updating consists of adding a new file or module to an installed application, or replacing or removing an installed application, file, or module. After replacement of a full application, the old application is uninstalled. After replacement of a module, file, or partial application, the old installed module, file, or partial application is removed from the installed application.

There are several ways to update application files. You can update enterprise applications or modules using the administrative console or a **wsadmin** tool. Both ways provide identical updating capabilities. Further, in some situations, you can update applications or modules without restarting the application server.

Note that WebSphere Application Server V6 supports J2EE 1.4 enterprise applications and modules. If you are deploying J2EE 1.4 modules, ensure that the target server and its node is running WebSphere Application Server V6. You can deploy J2EE 1.4 modules to WebSphere Application Server V6 servers or to clusters that contain such cluster members only. You cannot deploy J2EE 1.4 modules to servers on nodes running prior versions of WebSphere Application Server or to clusters that contain cluster members running such earlier versions.

There are several options to update the applications files deployed on a server or cluster:

- Administrative console update wizard

  Use this option to update enterprise applications, modules, or files already installed on a server.

  - Use full application to update an EAR file.
  - Use single module to update a WAR, EJB JAR, or connector Resource Adapter Archive (RAR) file.
  - Use a single file to update a file other than an EAR, WAR, EJB JAR, or RAR file.
  - Use partial application to update or remove multiple files.

- The **wsadmin** scripts

  The **wsadmin** tool supports two scripting languages: Jacl and Jython. Five objects are available when you use scripts:

  - `AdminControl`: Use to run operational commands.
  - `AdminConfig`: Use to run configurational commands to create or modify WebSphere Application Server configurational elements.
  - `AdminApp`: Use to administer applications.
  - `AdminTask`: Use to run administrative commands.
  - `Help`: Use to obtain general help.

  The scripts use these objects to communicate with MBeans that run in WebSphere Application Server processes. MBeans are Java objects that represent Java Management Extensions (JMX) resources. JMX is an optional package addition to Java 2 Platform Standard Edition (J2SE). JMX is a technology that provides a simple and standard way to manage Java objects.

- Using administrative programs (JMX)

  Update an application in the following ways:

  - Update the entire application.
  - Add to, update or delete multiple files in an application.
  - Add a module to an application.
  - Update a module in an application.
  - Delete a module in an application.
  - Add a file to an application.
  - Update a file in an application.
  - Delete a file in an application.

- WebSphere rapid deployment

  WebSphere rapid deployment offers the following advantages:

  - Create a new J2EE application quickly without the overhead of using an integrated development environment (IDE).
  - Package J2EE artifacts quickly into an EAR file.
  - Test and deploy J2EE artifacts quickly on a server.

  Refer to 13.3.3, "Application deployment" on page 390 for more information about the rapid deployment tool.

- Hot deployment and dynamic reloading

  Hot deployment and dynamic reloading requires that you directly manipulate the application or module file on the server where the application is deployed, that is, the new files are copied directly into the EAR directory on the relevant server or servers.

Refer to Chapter 12, "Planning for application deployment" on page 345 for further details about what application deployment options are available in WebSphere Application Server V6.

Changes in topology

The Deployment Manager node contains the master configuration files. Each node has its required configuration files available locally. Configuration updates should be done on the Deployment Manager node. The Deployment Manager process then synchronizes the update with the node agent through the Filetransfer application. File synchronization is a one-way task, from the Deployment Manager to the individual nodes. Changes made at node level are temporary and will be overridden by the master configuration files at the next file synchronization. If security is turned on, HyperText Transport Protocol - Secure (HTTPS) is used instead of HTTP.

File synchronization

File synchronization settings are customizable by cell. Each cell can have distinct file synchronization settings. File synchronization can be automatic or manual.

- Automatic

  Automatic synchronization can be turned on using admininstrative clients. The default file synchronization interval is 60 seconds and starts when the application server starts.

- Manual

  Manual synchronization can be performed using the administrative console, the **`wsadmin`** tool or using the **`syncNode`** command line tool provided in the <installation_root>\bin directory.

The file synchronization process should coincide with the whole change management process. In general, it is recommended to define the file synchronization strategy as part of the change management process.

13.2.5 Problem management

The most significant change within WebSphere Application Server V6 troubleshooting support and serviceability is the enhanced ability to automatically detect and recover from problems. In addition, a number of smaller changes add up to improved ability to solve problems quickly.

Improved message text, new message prefixes for key product components used during installation, migration, and initial configuration have been improved. Additional components now have predefined messages. Message prefixes have changed.

J2SE 1.4 introduced the Java logging framework, java.util.logging. WebSphere Application Server ensures that messages and trace generated via both JRAS is handled by the java.util.logging infrastructure. Thus, java.util.logging handlers connected to the root logger are able to receive all WebSphere Application Server log content.

Logging levels can be controlled via the administrative console's Troubleshooting section. WebSphere Application Server also builds its logs from the java.util.logging framework by connecting its handlers to the root logger.

The java.util.logging logging infrastructure allows for flexible pluggability of custom handlers to enable custom logs. By appropriate configuration, the handlers can receive WebSphere Application Server's logged events, as well as events from application loggers.

In WebSphere Application Server V6, the use of the JRAS API is deprecated. You should use the java.util.logging infrastructure instead.

Common base events are data structures used to describe situations that occur within the system. Common base events are used for various purposes, including representing things such as business events, configuration events, error events, and so on. WebSphere Application Server now uses common base events as the internal representation of logged messages.

Thread names can be included in logs. Thread name has been added to the advanced log format and Log Analyzer format. The advanced log format is available as an output format for the trace log and system out log. The thread name is now included in this format to enable easier correlation with other types of diagnostic data. The Log Analyzer format is available as an output format for the trace log. The thread name is now included in this format to enable easier correlation with other types of diagnostic data.

Best practices for the run-time environment

A proactive approach to problem management is always the best. You should check the following to avoid issues with the run-time environment:

- Check that you have the necessary prerequisite software up and running.
- Check that they have the proper authorizations and that the definitions are correct.
- Check for messages that signal potential problems. Look for warnings and error messages in the following sources:
  - SYSLOG from other subsystems and products, such as TCP/IP, Resource Access Control Facility (RACF®), and so forth.
  - WebSphere Application Server error log.
  - SYSPRINT of the WebSphere Application Server for z/OS.
  - Component trace output for the server.

- Check the ports used by WebSphere Application Server. The ports that WebSphere Application Server uses must not be reserved by any other system component.
- Check that enough disk space for dumpfiles is available.
- Check your general environment:
  - System memory
  - Heap size
  - System has enough space for archive data sets
- Make sure all prerequisite fixes have been installed; a quick check for a fix can save hours of debugging.

Problem determination tools

WebSphere Application Server V6 contains a number of tools to aid in problem determination. These tools allow you to quickly collect key information.

- The collector tool

  The collector tool gathers extensive information about your WebSphere Application Server installation and packages it in a JAR file that you can send to IBM Customer Support to assist in determining and analyzing your problem. Information in the JAR file includes logs, property files, configuration files, operating system and Java data, and the absence or level of each software prerequisite.

- Log Analyzer

  The Log Analyzer takes one or more service or activity logs, merges all the data, and by default, displays the entries in unit of work groupings. It analyzes event and error conditions in the log entries to provide message explanations.

  By default, the Logs pane of the Log Analyzer displays log entries by unit of work. It lists all the unit of work instances and its associated entries from the logs that you have opened. You might find the unit of work grouping useful when you are trying to find related entries in the service or activity log or when you are diagnosing problems across multiple machines.

- dumpNameSpace

  The dumpNameSpace tool is used to dump the contents of a name space accessed through a name server. The dumpNameSpace tool is based on JNDI. When you run the dumpNameSpace tool, the naming service must be active. The dumpNameSpace tool cannot dump name spaces local to the server process, such as those with java: and local: URL schemes.

  The tool dumps the server root context for the server at the specified host and port unless you specify a non-default starting context which precludes it. The tool does not dump the server root contexts for other servers. The tool runs from a command line or using its program interface. If you run the dumpNameSpace tool with security enabled, a login prompt is displayed.

Logs and traces

Log files and traces should be properly named. It is recommended that you name log files according to the application that they belong to and group them in different directories. Log files should be cleaned periodically (saved to a media and then deleted). WebSphere Application Server can write system messages to several general purpose logs. These include:

- JVM logs

  The JVM logs are written as plain text files. They are named SystemOut.log and SystemErr.log and are located in the following location:

  <installation_directory>/profiles/<profile_name>/logs/<server_name>

  There are two ways to view the JVM logs for an application server:

  – Use the administrative console. This supports viewing the JVM logs from a remote machine.

  – Use a text editor on the machine where the logs are stored.

- Process logs

  WebSphere Application Server processes contain two output streams that are accessible to native code running in the process. These streams are the standard output (stdout) and standard error (stderr) streams. Native code, including the JVM, can write data to these process streams.

  By default, the stdout and stderr streams are redirected to log files at application server startup, which contain text written to the stdout and stderr streams by native modules (Dynamic Link Libraries (DLLs), Executables (EXEs), UNIX system libraries (SO), and other modules).

  By default, these files are stored as:

  – <installation_root>/logs/<applicationServerName>/native_stderr.log
  – <installation_root>/logs/<applicationServerName>/native_stdout.log

- IBM service log (activity.log)

  The service log is a special log written in a binary format. You cannot view the log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log.

You can view the service log in two ways:

- It is recommended that you use the Log Analyzer tool to view the service log. This tool provides interactive viewing and analysis capability that is helpful in identifying problems.
- If you are unable to use the Log Analyzer tool, you can use the Showlog tool to convert the contents of the service log to a text format that you can then write to a file or dump to the command shell window.

IBM service log is located in the following directory:

<installation_root>/logs

Fix management

You should have a fix management policy in place, and you should test fixes before you apply them to the production system. For available fixes, see the WebSphere Application Server support page:

http://www.ibm.com/software/webservers/appserv/was/support/

13.3 Planning for application administration

There are many key elements to WebSphere Application Server V6 application administration. The application environment must be configured carefully to ensure operational efficiency. Setting up and configuring the application environment properly avoids many issues. The native system applications need to be understood too. Although they cannot be altered, their impact on the environment needs to be noted. There are also many considerations for deploying an application. The new rapid deployment technologies have simplified this process greatly.

13.3.1 Application environment

A great deal of planning needs to happen before the applications are installed into the environment. An understanding of the process beforehand is essential to success.

Previous chapters discussed in detail the planning activities for infrastructure, topology selection, installation, and configuration. The following list summarizes the most relevant activities.

You can use this list as a quick checklist for steps validation:

- Plan for the infrastructure.
- Understand topologies.

- Plan the WebSphere Application Server V6 installation.
  - Select the topology.
  - Review hardware and software prerequisites.
  - Install components.
  - Create profiles.
  - Configure ports.
  - Configure security.

These topics are covered in detail in the following chapters:

- Chapter 7, “Planning for infrastructure” on page 189
- Chapter 8, “Topology selection criteria” on page 219
- Chapter 9, “Topologies” on page 247
- Chapter 10, “Planning for installation” on page 285

13.3.2 System applications

System applications are J2EE enterprise applications that are central to a WebSphere Application Server product, such as the administrative console and file transfer application. System applications are no longer shown in the list of installed applications on the console Enterprise Applications page, to prevent users from accidentally stopping, updating or removing them.

Because system applications are an important part of a WebSphere Application Server product, system applications are deployed when the product is installed and are updated only through a product fix or upgrade. Users cannot change the metadata for system applications such as J2EE bindings or J2EE extensions. Metadata requiring a change must be updated through a product fix or upgrade.

13.3.3 Application deployment

After the environment is installed and configured, it is time to deploy the application. The WebSphere Application Server V6 setup wizard creates the necessary components for the application to be deployed. The default installation can be used or customized as needed.

You can run the rapid deployment batch tool to create, configure, and launch rapid deployment projects using the WebSphere Application Server V6 runtime environment. The rapid deployment tool is a non-graphical user interface mode that runs from a command line. There are two ways to configure the monitored directory, each performing separate and distinct tasks. You can specify the monitored directory as a free-form project or an automatic installation project.

Free-form project

The rapid deployment tools free you from understanding J2EE application structure, as well as having to update and modify basic deployment descriptor information, such as new servlet or EJB entries. With the free-form approach, you can place in a single project directory the individual parts of your application, such as Java source files that represent servlets or enterprise beans, static resources, XML files, and other supported application artifacts. The rapid deployment tool then uses the artifacts that are dropped in this free-form project directory to place them in the appropriate J2EE project structure automatically and to generate any additional required artifacts to construct a J2EE-compliant application and deploy that application on a target server.

The rapid deployment tools detect additions or changes to J2EE artifacts in the free-form project and run a build. As you drop files into the free-form project, the necessary J2EE projects are created, if these projects do not already exists. For example, if you drop a servlet source file, a Web project and an application project is created. The rapid deployment tool then maps the artifacts to their correct location in the appropriate project. In this case, the servlet source file is compiled and its compiled class files are copied to the following directory:

```
<war_root>\WebContent\WEB-INF\classes
```

The rapid deployment tool then generates or updates any additional required deployment artifacts. In this case, the web.xml file for the Web project is updated to include a servlet and servlet mapping entry.

The rapid deployment tool publishes the updated application on the server. If the target server is a local (on the same machine running the rapid deployment session), the server runs the resources from the workspace without creating an EAR file. However, if you choose to use the optional earExportPath option that specifies the directory location of the output file, an EAR file is created. If the target server is remote, it generates an EAR file and transfers the file to the remote server.

Benefits

The benefits of using the free-form approach is that you can focus on the structure of your application independent of the requirements of J2EE or other standards. One of the complexities of J2EE development is that it defines the location of artifacts in an application relative to one another. For example, within a Web application, class files must be placed in the directory WEB-INF/classes and JSP files must be placed outside of the WEB-INF folder. This structural requirement must be understood and followed during development in order for the application to operate correctly. The rapid deployment tools eliminates this structural requirement, allowing the source project to follow your development defined structure and freeing you from having to understand the J2EE-defined structures.

Automatic installation project

The rapid deployment tools simplify the process of installing, modifying, and uninstalling applications on a server. Automatic installation relies on a single project directory that listens for J2EE compliant applications (EAR files) or modules (WAR, JAR, or RAR files). The rapid deployment tool then deploys the artifacts to the target server.

Adding, updating, and removing enterprise archives

You can place EAR files in the automatic application installation project. The EAR file is deployed automatically to the server by expanding into the installedApps folder of the local WebSphere Application Server. When you delete EAR files from the automatic installation project, the application is uninstalled from the server. You can place a modified version of the same EAR file in the automatic installation project, and the application is reinstalled.

Adding, updating, and removing modules

You can place WAR, RAR, or EJB jar files in the automatic installation project. A supporting EAR wrapper is generated in the gen subdirectory, and that EAR file then is deployed to the server by expanding into the installedApps folder of the local WebSphere Application Server. For RAR files, a wrapper is not created. The stand-alone RAR files are simply published to the server. When you delete module files from the automatic application installation project, the application is uninstalled from the server. In addition, any supporting EAR wrappers that were generated in the `gen` subdirectory are removed. You can place a modified version of the same module file in an automatic application installation project, then the application is reinstalled. In addition, any supporting EAR wrappers that were generated in the gen subdirectory are modified.

Benefits

The benefit of using the automatic installation project is that it frees you from the complexity of administering your application. You can use **`wsadmin`** or the administrative console to install the application through a multi-step wizard. For further details the application deployment options, refer to Chapter 12, “Planning for application deployment” on page 345.

13.4 Plan for backup and recovery

In general, computer hardware and software is very reliable, but sometimes failures can occur and damage a machine, network device, software product, configuration, or more importantly, business data. It is important to plan for such occurrences. Much of the planning concerns elements are external of WebSphere Application Server V6 but still need to be considered in order to provide seamless operation.

There are a number of stages to creating a backup and recovery plan, which the following sections discuss.

13.4.1 Risk analysis

The first step to creating a backup and recovering plan is to complete a comprehensive risk analysis. The goal is to discover which areas are the most critical and which hold the greatest risk. It is also important to identify which business processes are the most important and how they will be effected by application outages.

13.4.2 Recovery strategy

When the critical areas have been identified, you need to develop a strategy. There are numerous backup and recovery strategies that are available which vary in recovery time and cost. In most cases, the cost increases as the recovery time decreases. The key to developing the proper strategy is to find the proper balance between recovery time and cost. The business impact is the determining factor in finding the proper balance. Business critical processes need quick recovery time to minimize business losses. Therefore, the recovery costs are greater.

13.4.3 Backup plan

With a recovery strategy in hand, a backup plan needs to be created to handle the daily backup operations. There are numerous backup methods varying in cost and effectiveness. A *hot backup site* provides real-time recovery by automatically switching to a whole new environment quickly and efficiently. Because of the cost, only the largest sites use hot backup. For less critical applications, *warm* and *cold backup sites* can be used. These are similar to hot backup sites, but are less costly and effective. More commonly, sites use a combination of backup sites, load-balancing, and high availability.

More common backup strategies combine replication, shadowing, and remote backup as well as the more mundane methods such as tape backup or

Redundant Array of Independent Disks (RAID) technologies. All of which are just as viable as a hot backup site but require longer restore times. Any physical backup should be stored at a remote location in order to be able to recover from a disaster, such as a fire. New technologies make remote electronic vaulting a viable alternative to physical backups and many third party vendors offer the service.

Backup and restore configuration

The WebSphere Application Server configuration, stored in the directory <install_root>/config, can be backed up to a zipped file by using the **`backupConfig`** utility.

For a stand-alone node, run the **`backupConfig`** utility at the node level. For a network deployment cell, run the **`backupConfig`** utility at the Deployment Manager level, because it contains the master repository. Do not perform **`backupConfig`** at the node level of a cell.

13.4.4 Recovery plan

If a disaster does occur, a plan for restoring operations as efficiently and quickly as possible must be in place. The recovery plan must be coordinated with the backup plan to ensure the recovery happens smoothly. The appropriate response must be readily available so that no matter what situation occurs, the site will be operational at the agreed upon disaster recovery time. A common practice is to rate outages from minor to critical and have a corresponding response. For example, a hard disk failure could be rated as a Class 2 outage and have a Class 2 response where the disk gets replaced and replicated with a 24 hour recovery time. This makes recovering easier because resources are not wasted and disaster recovery time is optimized.

Another key point of the recovery plan must address what happens after the recovery. Minor disasters, such as disk failure, have little impact afterwards but critical disasters, such as the loss of the site, have a significant impact. For example, if a hot backup site is used, the recovery plan must account for the return to normal operation. New hardware or possibly a whole new data center might need to be created. Post-disaster activities need to be completed quickly to minimize costs.

The **`restoreConfig`** command restores the configuration of your stand-alone node or cell from the zipped file that was created using the **`backupConfig`** command.

13.4.5 Update and test process

You must revise the backup and recovery plan on a regular basis to ensure that the recovery plan meets current needs. You also need to test the plan on a regular basis to ensure that the technologies are functional and that the personnel involved know their responsibilities. In addition to the regular scheduled reviews, the backup and recovery plan should be reviewed when new hardware, technologies, or personnel are added.

13.5 Plan for availability

In today's Internet environment, it is crucial that your data and applications be available to you when you need them. If your customers cannot access your Web site because your server is down, they can go to your competitors instead.

Availability is the measure of how often your data and applications are ready for you to access when you need them. Different companies have different availability needs. Different servers within the same company can have different availability needs. It is important to note that availability requires detailed planning.

13.5.1 Availability concepts

Before you can plan for the availability it is important for you to understand some of the basic availability concepts. Availability is measured in terms of outages, which are periods of time when the application is not available to users. During a planned, or scheduled, outage, you deliberately make your system unavailable to users. You might use a scheduled outage to perform backup activities or apply fixes.

An unplanned, or unscheduled, outage is usually caused by a failure. You can recover from some unplanned outages, such as disk failure, system failure, power failure, program failure, or human error, if you have an adequate backup and recovery strategy. However, an unplanned outage that causes a complete system loss, such as a natural disaster or a fire, requires you to have a detailed disaster recovery plan in place in order to recover.

Your backup window is the amount of time that your server can be unavailable to users while you perform your backup operations. Your backup window is a scheduled outage that usually occurs in the night or on a weekend when your server has less traffic.

There are several levels of availability. You should:

- Estimate the value of availability.
- Decide what level of availability you need.

There are several different ways of approaching availability, based on your setup and the types of outages that you are anticipating, as follows:

- Maintain availability solutions for a single server.
- Prevent unplanned outages.
- Shorten unplanned outages.
- Shorten planned outages.
- Have availability solutions using multiple servers.

If you require a level of availability higher than what you can achieve with a single server implementation then you should consider clusters. Clusters can help provide nearly full availability for your critical applications and data. For more details about clusters, see Chapter 9, “Topologies” on page 247.

Note: Take care when measuring availability in percentages. An e-commerce site that crashes, looses all system data, and is down for one minute every 10 minutes can be said to be available 90% of the time. However, the site is practically unusable most of the time, because a typical shopping process takes more than 10 minutes to complete.

13.5.2 New high availability features

The WebSphere Application Server high availability framework eliminates single points of failure and provides peer-to-peer failover for applications and processes running within the application server. The WebSphere Application Server high availability framework also allows integration of WebSphere Application Server into an environment that might be using other high availability frameworks, such as High Availability Cluster Multi-Processing (HACMP) to manage non-WebSphere Application Server resources.

High availability manager

The high availability manager is responsible for running key services on available application servers rather than on a dedicated server, such as the Deployment Manager. It takes advantage of fault tolerant storage technologies such as network attached storage (NAS), which significantly lowers the cost and complexity of high availability configurations. It also offers hot standby and peer failover for critical services.

The high availability manager monitors the application server environment and provides peer-to-peer failover of application server components that are part of a core group. If an application server component fails, the high availability manager takes over the in flight and in doubt work for the failed server. This action significantly improves application server uptime. In a carefully planned environment, it can provide a 99.999% application server availability expectancy rate.

All components of an environment managed by the high availability manager are members of a core group. WebSphere Application Server V6 provides a default core group that is created when the product is installed. When new server instances are created, they are added automatically to this default core group. Additional core groups can be created. However, for most environments, one core group is usually sufficient and additional core groups should not be added unless there is a specific need to do so. If your environment has more than one core group, use access point groups to enable the core groups to communicate with each other. The core groups that communicate can be in the same cell or in different cells.

Data replication service

Replication is a service that transfers data, objects, or events among application servers. Data replication service (DRS) is the internal WebSphere Application Server component that replicates data. The session manager uses the data replication service when configured to do memory-to-memory replication. When memory-to-memory replication is configured, the session manager maintains data about sessions across multiple application servers, preventing the loss of session data if a single application server fails.

Dynamic cache uses the data replication service to further improve performance by copying cache information across application servers in the cluster, preventing the need to repeatedly perform the same tasks and queries in different application servers.

Stateful session beans use the replication service so that applications using stateful session beans are not limited by unexpected server failures.

You can define the number of replicas that the data replication service creates on remote application servers. A replica is a copy of the data that copies from one application server to another. The number of replicas that you configure affects the performance of your configuration. Smaller numbers of replicas result in better performance because the data does not have to be copied many times. However, if you create more replicas, you have more redundancy in your system. By configuring more replicas, your system becomes more tolerant to possible failures of application servers because the data is backed up in several locations.

By having a single replica configuration defined, you can avoid a single point of failure in the system. However, if your system must be tolerant to more failure, you most introduce extra redundancy in the system. Increase the number of replicas that you create for any HTTP session that is replicated with the data replication service. Any replication domain that is used by dynamic cache must use a full group replica.

Session manager, dynamic cache, and stateful session beans are the three consumers of replication. A consumer is a component that uses the replication service. When you configure replication, the same types of consumers belong to the same replication domain. For example, if you are configuring both session manager and dynamic cache to use data replication service to replicate objects, create separate replication domains for each consumer. Create one replication domain for all the session managers on all the application servers and one replication domain for the dynamic cache on all the application servers. The only exception to this rule is to create one replication domain if you are configuring replication for HTTP sessions and stateful session beans. Configuring one replication domain in this case ensures that the backup state information is located on the same backup application servers.

High availability implementation is covered in more in detail in Chapter 14, “Planning for performance, scalability, and high availability” on page 405.

13.6 Plan for performance

Planning for performance requires that you set performance objectives, create benchmarks that are based on those objectives, and make system growth plans.

When planning your system's performance, you need to fully understand the business requirements that your system is addressing and be able to translate those business needs into performance objectives. These performance objectives are then formulated into non-functional performance requirements.

Perhaps the best way to start is to estimate the maximum hourly and daily throughput required during peak business periods. After that, you can decide what average response time is acceptable.

13.6.1 Monitoring

The first step in understanding the performance of your system is to monitor the key elements mentioned in this section. Knowing what elements have the greatest impact and how they effect the system in general is very important. They can easily be divided into two categories: system resources and application resources.

System resources

System resources are elements of the operating system. They show the overall health of the server. Issues with the system resources are symptomatic of application problems that might not be apparent monitoring just the application. Typical system resources are:

- Processor: depicts overall system workload
- System memory: available physical system memory
- Virtual memory: available virtual system memory
- Disk space: available hard disk space
- Paging: amount of activity write/read from disk drives

Application resources

The health of the application itself can be seen by monitoring the application resources. Typical application resources to monitor are:

- Average response time: amount of time to complete a transaction
- Number of requests: number of transactions completed
- Number of live HTTP sessions: concurrent user sessions
- Pools: the Web server and EJB thread pools, database and connection pool size
- JVM heap size: amount of memory allocated to an application
- Garbage collection: java memory collection
- Java memory: WebSphere application memory
- Application objects: over-utilization
- Memory leaks: uncollected application memory

Performance monitoring tools

There are many tools to monitor the performance of the system. System resources can be monitored with operating system specific tools. Consult the operating system help to find information about which tools are available. To monitor the application resources, there are WebSphere Application Server specifics tools that ship with WebSphere Application Server V6. The most common of these is the Tivoli Performance Viewer.

Tivoli Performance Viewer

The Tivoli Performance Viewer enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the WebSphere Application Server administrative console. When the Tivoli Performance Viewer is running in a single server environment, the collection and viewing of the data occurs in the same JVM. Because the collection and viewing of data occurs in the application server, performance is affected by the various user and logging settings.

When the Tivoli Performance Viewer is running in a network deployment environment, the data is collected at each of the nodes and stored in the node agent's memory. Data can then be viewed from the Deployment Manager. This architecture enables the monitoring work to be distributed among the nodes. Similar to the single server environment, the various user and logging settings directly influence the extent to which performance is affected.

Performance Monitoring Infrastructure

The Performance Monitoring Infrastructure (PMI) is the core monitoring infrastructure for WebSphere Application Server. The performance data provided by PMI helps to monitor and tune the application server performance. PMI provides a comprehensive set of data that explains the runtime and application resource behavior. For example, PMI provides database connection pool size, servlet response time, EJB method response time, JVM garbage collection time, processor usage, and so on. This data can be used to understand the runtime resource utilization patterns of the thread pool, connection pool, and so on, and the performance characteristics of the application components such as servlets, JSPs, and EJBs.

WebSphere Application Server V6 performance is covered more in detail in Chapter 14, "Planning for performance, scalability, and high availability" on page 405.

13.7 Plan for security

When planning security for WebSphere Application Server V6, it is important to have a comprehensive security policy. The security for WebSphere Application Server should coordinate neatly with the overall environment security. WebSphere Application Server V6 adheres to standard J2EE specifications as closely as possible and integrates easily with existing security systems. There is no single fix for security concerns. However, proper planning and diligence can keep systems functional and minimize the impact on business.

Security can be divided into the following areas:

- Physical security

  Physical security encompasses the area where the environment is located and the major concern is access to the site. Commonly, such areas are physically secured and access is limited to a small number of individuals

- Logical security

  Logical security is the software used to protect the various systems. Password protection is the most common logical security but can also include firewalls and protective devices.

- Application security

  Application security is the use WebSphere Application Server V6 technologies to secure the application from intrusion. WebSphere Application Server V6 provides many plug-in points to integrate with enterprise software components to provide end-to-end security.

Security policy

There are a number of key principles of a security policy:

- Identify key assets

  Identify critical areas of business and those assets that host them. There are many methods to implement a security plan. By identifying those key assets, you can adopt which methods are best for the environment and create an effective security policy.

- Identify vulnerabilities

  Complete a comprehensive assessment of the environment to identify all the threats and vulnerabilities. Examine each and every area of the environment to find any way the system can be compromised and it is very important to be thorough. It is important to remember to examine the three types of security mentioned earlier: physical, logical, and application. This can be a very resource intensive activity but it is crucial to the security of the environment.

- Identify acceptable risk

  After completing a vulnerability assessment, the acceptable risk must be determined. In most instances, this will be a cost issue. To completely secure an environment would be extremely expensive, so compromises have to be made to complete the security policy. In most cases, the most cost effective method to meet the required security level will be used. For instance, in a system that contains mission critical data for a company, the most advanced firewall available would be necessary. However, on a test system which had no external access, the appropriate security level could be met with minimal or no firewalls.

- Layered security model

  In complex systems, it is important to have multiple layers of security to ensure the over-all safety of the environment. A layered security model plans for expected loss and minimizes the impact. It also ensures that all components are protected, from the user to the backend data systems, and a failure in any does not impact the whole environment. Refer to 13.7.1, “Layered security architecture” on page 402 for more details.

Security configuration

After you have created the security policy, you must implement it. You have to implement steps to configure the physical, logical, and application security as recommended in the security policy. In many cases, recommended configurations will not be possible and, once again, compromises must be made. Any discrepancies should be noted so that they can be addressed and possibly remedied at a future date.

Security reviews

A timely and regular review of the security policy is essential to its effectiveness. As the environment evolves over time, the security policy must also evolve. Regular appraisals of the security policy, including key assets, vulnerability assessment, and acceptable risk are needed to maintain the expected level of security.

13.7.1 Layered security architecture

There are several communication links from a browser on the Internet, through Web servers and application servers, to the enterprise data at the back-end. WebSphere Application Server security is built on a layered security architecture. The building blocks that comprise the operating environment for security within the WebSphere Application Server are:

- Operating system security

  The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. These services include the file system security support that secure sensitive files in the product installation for WebSphere Application Server, such as the configuration files. The system administrator can configure the product to obtain authentication information directly from the operating system user registry.

- Network security

  The network security layers provide transport level authentication and message integrity and encryption. You can configure the communication between separate application servers to use SSL and HTTPS. Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.

- JVM 1.4 security

  The JVM security model provides a layer of security above the operating system layer.

- Java 2 security

  The Java 2 security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.

- Common Security Interoperability security

  Any calls made among secure Object Request Brokers (ORB) are invoked over Object Management Groups Common Security Interoperability Version 2 (OMG CSIv2) security protocol that sets up the security context and the necessary quality of protection. After the session is established, the call is passed up to the EJB layer. For backward compatibility, WebSphere Application Server supports the Secure Authentication Service security protocol, which was used in prior releases of WebSphere Application Server and other IBM products.

- J2EE security

  The security collaborator enforces J2EE-based security policies and supports J2EE security APIs.

- WebSphere security

  WebSphere Application Server security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere Application Server security technologies and features to support the needs of a secure enterprise environment.

The various security layers have important implications and should be examined carefully. The implications should be integrated into the security policy to ensure safety.

13.8 Summary

The system administrator must master many concepts in order to manage the environment efficiently. In many ways, managing a WebSphere Application Server V6 environment is similar to managing earlier versions but the new features of WebSphere Application Server V6 must also be mastered. The same methods and processes used with previous versions can easily be adapted to the new features. Proper planning can ease the transition and avoid common issues.

An in-depth knowledge of the functionality is needed to properly plan for system management. A proactive approach to such areas as system administration, application administration, backup and recovery, high availability, security, and performance can ease daily operations. Common processes, such as change management and problem determination, are applicable. However, new functionality alters how these processes are completed, and thus, the processes themselves need to be modified. Understanding the functionality of WebSphere Application Server V6 in advance can aid in the development of plans for system management.

14

Planning for performance, scalability, and high availability

This chapter discusses at a high level the items to consider when implementing WebSphere Application Server V6 so that the environment performs well and is highly scalable. It contains the following sections:

14.1 Key concepts and terminology

This section covers some fundamental concepts and terms that are related to performance an scalability planning. These terms are not necessarily unique to WebSphere, but are more general in terms of how you would think about a system deployment.

14.1.1 Scalability

Scalability, in general terms, means adding hardware to improve performance. However, you should not simply add hardware without doing some investigation and possible software tuning first to identify potential bottlenecks or any other performance related software configurations. Adding more hardware might not necessarily improve the performance if the software is not tuned correctly. When the software optimization has been done, then the hardware resources should be considered as the next step for improving performance. There are two different ways to improve performance when adding hardware: vertical scaling and horizontal scaling.

Vertical scaling

Vertical scaling means increasing the throughput of the site by handling more requests in parallel. From a hardware perspective, we can increase the number of the processors for the server. For example, by upgrading the system hardware with two or more processors, the server can potentially gain twice the processing capacity. Thus, more requests should be able to be handled at the same time. This concept is relatively easy to implement and can apply to any server in the system. Vertical scaling should be examined at every potential bottleneck tier.

Horizontal scaling

Horizontal scaling means adding duplicated servers to handle additional load. This applies to multi-tier scenarios and can require the use of a load balancer that sits in front of the duplicated servers, so that the physical number of servers in your site are hidden from the Web client.

14.1.2 Throughput

Throughput means number of requests relative to some unit of time the system can process. For example, if an application can handle 10 customer requests simultaneously and each request takes one second to process, this site can have a potential throughput of 10 requests per second. Let us say that each customer on average submits 60 requests per visit, then we can also represent throughput by estimating six visits per minute or 360 visits per hour.

14.1.3 Response Time

Response time is the time it takes when the user initiates a request at the browser until the result of the HTML page returns to the browser. At one time, there was an unwritten rule of the Internet known as the *eight second rule*. This rule stated that any page that did not respond within eight seconds would be abandoned. Many enterprises still use this as the response time benchmark threshold for Web applications.

14.1.4 Workload Management

Workload management (WLM) is a WebSphere facility to provide load balancing and affinity between nodes in a WebSphere clustered environment. WLM can be an important facet of performance. WebSphere uses WLM to send requests to alternate members of the cluster if the current member is too busy to process the request in a timely fashion. WebSphere will also route concurrent requests from a user to the same application, as EJB calls, and session state are in memory on the application server that answered the previous request.

14.1.5 Availability

Also known as resiliency, availability is the description of the system's ability to respond to requests no matter the circumstances. Both performance and scalability can affect availability. The key is to determine the threshold for availability. The most common method of describing availability is by the "nines" or the percentage availability for the system. 99.9% availability represents 8.5 hours of outage in a single year. To add an additional 9, or 99.99% represents approximately 1 hour of outage in a single year. The cornerstone measurement of "five nines" or 99.999% availability represents an outage of less than five minutes in a year. Calculating availability is a simple process using the formula expressed in the Example 14-1, where MTBF is the mean time between failure and MTTR is the mean time to recovery.

Example 14-1 Availability calculation formula

```
Availability = (MTBF/(MTBF + MTTR)) X 100
```

When planning for performance and scalability, you should take availability into consideration. Make sure, however, that the business case justifies the costs. In many real world examples, moving a system availability from 99.9% to 99.99% could be extremely expensive. It can also be true that the system will only be used during regular business hours on regular working days. This implies that an availability of 99.9% would be more than adequate to meet the operational window.

14.1.6 Maintainability

Maintainability is the ability to keep the system running before, during, and after scheduled maintenance. When considering maintainability in performance and scalability, remember that maintenance need to be periodically performed on hardware components, operating systems, and software products in addition to the application components. Maintainability allows for ease of administration within the system by limiting the number of unique features found in duplicated resources. There is a delicate balance between maintainability and performance. It is always a distinct possibility that the goal of maintainability will clash with performance. The ideal is that a high performing system is easily maintained, and that is still what you would strive for.

14.1.7 Session state

The nature of running Web applications is such that there is the need to maintain the state of the user within the application between requests. Use of the session information, however, walks a fine line between convenience for the developer and performance and scalability of the system. It is not practical to eliminate session data altogether, but care should be taken to minimize the amount of session data passed. Persistence mechanisms will decrease capacity of machines in the system, or incur additional costs to increase the capacity or even the number of servers. Care should be taken when designing your WebSphere environment to take session needs into account as early as possible.

14.1.8 WebSphere Application Server security

When discussing WebSphere Application Server security, there are many layers to discuss. There is SSL at the front end tier, SSL in the plug-in, method security, and back end authentication mechanisms. Each of these layers can present performance and scalability issues.

SSL at the front end or Web tier is a common implementation. This allows for secured purchases, secure viewing of bank records, and many other examples. SSL handshaking, however, is expensive from a performance perspective. The Web server is responsible for encrypting data sent to the user, and decrypting the request from the user. If a site will be operated using SSL more often than not, it might make sense to implement some form of SSL accelerator hardware in the server, or even an external device that offloads all SSL traffic prior to communication with the Web server.

SSL in the plug-in is also used, though not as common. This implementation allows the Web server to pass communication to the application server using https instead of http. In some cases, this is due to security constraints placed on the business. In others, this is due to isolation of security realms. Depending on the volume of transactions, additional hardware accelerators or appliances can be used to increase performance.

Method security is the design of using WebSphere Application Server integrated security to restrict access to specific methods or servlets within the application. This is an effective tool in practice, but it can create performance issues if too many rules are imposed. There are no hard and fast rules associated with this security implementation, but be aware that calls to an external LTPA provider (in a clustered environment) can create performance issues if every method is secured.

Backend authentication mechanisms refer to using an LTPA provider such as LDAP to authenticate users for the system. This is a standard practice, and by itself does not represent a performance issue. What can become a performance or scalability issue, however, is understating the potential load on the LDAP server. When planning for a secured environment, pay special attention to the sizing and implementation of the directory server. Also, pay attention that this directory does not introduce a new single point of failure. Losing the authentication mechanism can definitely impact site performance.

For more details about security, refer to Chapter 17, “Planning for security” on page 485.

14.2 Scalability

Adding resources usually affects the dynamics of the system and can potentially shift the bottleneck from one resource to another. In a single tier scenario Web, Application, and Database servers are all running in the same system. Creating a cluster by adding a separated system running another Application server should improve the throughput. But at the same time, this server introduces new communication traffic to the Database server. How much network bandwidth will this server configuration consume? What will be the performance improvement by adding more servers?

Because of these unknowns, it is recommended that you always perform a scalability test to identify how well the site is performing after a hardware change. Throughput can be measured to ensure that the result meets your expectations.

You should also be aware that the law of diminishing returns does play a role when increasing scalability either vertically or horizontally. The law of diminishing returns is an economics principle which states that if one factor of production is increased while all others remain constant, the overall returns will reach a peak and then begin to decrease. This law can be applied to scaling in computer systems as well. This law means that adding two additional processors will not necessarily grant you twice the processing capacity. Nor will adding two additional horizontal servers in the Application Server tier necessarily grant you twice the request serving capacity. Additional overhead is introduced to manage those additional resources. While the degradation might be small, it is not a direct linear function of change in processing power to say that adding n additional machines will result in n times the throughput.

14.2.1 Scaling considerations

Important: Do not begin adding resources to the application environment until you know the environment has been properly tuned for the resources it currently uses.

Tuning

The first step in scaling considerations is to verify that the application and application environment has been adequately tuned. Tuning should be verified at the application code layer, the network layer, the server hardware layer, and the WebSphere Application Server layer.

Application code layer

The application code itself should be reviewed as part of the regular application life cycle to ensure it is using the most efficient algorithms and the most current APIs that are available for external dependencies. Take care to use optimized database queries or prepared statements instead of dynamic SQL statements.

Refer to the Chapter 11, “Planning for application development” on page 301 for discussions about development considerations.

Network layer

Tuning at the network layer is usually very cursory. Take the time to verify that port settings on the switches match the settings of the network interfaces. Many times, a network device is set to a specific speed, and the network interface is set to auto-detect. Depending on the operating system, this can cause performance problems due to the negotiation done by the system. Also, reviewing these settings establishes a baseline expectation as you move forward with scaling attempts.

Server hardware layer

Take the time to verify that the servers used in the application environment have been tuned adequately. This includes network options, TCP/IP settings, and even possibly kernel parameters. Verify disk configurations and layout as well. The tuning at this layer can be quite difficult without a specialist in the server hardware or the operating system, but the adjustments can lead to significant improvements in throughput.

Scaling

After you have verified and tuned the application, the next step is to examine the entire application environment to identify potential bottlenecks. Simply throwing additional resources into the environment without a fully understanding of what those resources are addressing could potentially worsen performance or scalability. It is necessary to have the end-to-end application environment knowledge to quickly identify the bottlenecks and ensure the appropriate response is takes timely.

When scaling at the network layer, such as with firewalls or switches, the most common solution is vertical scaling. Network devices have processing capacity, and use memory (RAM) much like any other hardware resource. Adding additional memory or processors to a network device will increase the throughput of that device, which positively impacts the scaling of that device. Moving from 10 MB to 100 MB or even 1 GB connections can definitely increase performance at the network layer.

When scaling at the Web server layer, the most often employed solution is horizontal scaling. This means adding additional Web servers to the environment. To do so, load balancers must be used. Be careful when adding servers to make sure that the load balancer has adequate capacity, or adding the new Web servers will simply shift the bottleneck from the Web tier to the load balancing tier. Vertical scaling can be performed as well by adjusting HTTP tuning parameters, or by increasing memory or processors.

Scaling at the Application Server layer can be done with either, or combination of, vertical or horizontal scaling. Vertical scaling at the application server layer can be done physically or logically. WebSphere Application Server is a Java application itself and can take advantage of additional processors or memory in the machine. WebSphere Application Server applications can be clustered vertically, providing multiple copies of the same application on the same physical machine. WebSphere Application Server also support horizontal cloning of applications across multiple machines, which do not necessarily need to be identical in terms of physical resources. A work load manager is required to do a smart load balancing across heterogeneous hardware resources.

At the data services layer, it is most common to implement vertical scaling. Adding multiple copies of the database can introduce complexities that can be unmanageable, and providing these resources to the application server might introduce higher costs than the value provided. A database server, however, can definitely make use of higher numbers of processors, or more memory. Most database management systems also can be tuned for performance with respect to I/O, pinned memory, and numbers of processors. Database tuning should definitely be asserted prior to adding additional resources, as these new resources might not improve the problem, although it is unlikely they can even increase the problem.

14.2.2 Application environment

Within the WebSphere Application Server V6 environment, there are many settings that can create increase application capacity and reduce performance issues. The purpose of this section is not to directly discuss those tuning parameters which are thoroughly covered in *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392. This section, however, should generate some thoughts as to settings that should be considered when designing a WebSphere Application Server V6 application.

Java Virtual Machine

When configuring a JVM, you should look at the minimum and maximum heap size closely. The system default for starting heap is 2 MB and 64 MB maximum heap size. Neither of these settings is likely desirable. Starting a JVM with that little memory means that the application must immediately switch context to allocate memory resources. This will slow down the execution of the application until it reaches the heap size it needs to run. A JVM that can grow too large does not perform garbage collection often enough, which can leave the machine littered with unused objects. The levels should be adjusted during testing to ascertain reasonable levels for both settings. You should also be aware that the prepared statement cache and dynamic fragment caching also consume portions of the heap, and as such, you might be required to make additional adjustments to the heap when those values are adjusted.

Web Container Thread Pool

The thread pool for the Web container should be closely monitored during initial load testing and implementation. This is the most common bottleneck in the application environment. Adjust the number of threads in the pool too high, and the system will spend too much time swapping between threads, and requests will not complete in a timely manner. Adjust the number of threads too low, and the Web server threads can back up and cause a spike in response at the Web server tier.

There are no hard rules in this space, as things like the type and version of Web server, and the percentage of requests that require the Application Server can impact the overall optimum tuning level. The best rule of thumb is to tune in small increments, and measure with identical loads over that time to make sure the adjustment hasn't crossed a threshold where the performance actually diminishes.

EJB container

The EJB container can also be another source of potential scalability bottlenecks. The inactive pool cleanup interval is a setting that determines how often unused EJBs are cleaned from memory. Set too low, and the application will spend more time instantiating new EJBs when an existing instance could have been reused. Set too high, and the application will have a larger memory heap footprint with unused objects remaining in memory. EJB container cache settings can also create performance issues if not properly tuned for the system.

The Performance Tuning guide within the InfoCenter and other publications, such as *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392, can provide some general rules for adjusting these values to increase the throughput or decrease response time of the system. You can access the WebSphere Application Server Information Center at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

Database connection pool

The database connection pool is another common location for bottlenecks, especially in data driven applications. The default pool size is usually 10, and depending on the nature of the application, and the number of requests, this pool can become full quite rapidly. During implementation testing, pay special attention to the pool usage, and adjust it higher as needed. This pool consumes additional Java heap as it gets used, so you might be required to go back and adjust the heap size after tuning the pool size. Also, be aware that in a clustered environment, the connection pool is a cell resource. This means that all nodes in the cluster share one pool. Adjust the pool in such way that it is large enough to handle the requests of all the servers participating in the cluster and cell.

14.2.3 Workload categorization

The workload defines how the performance of a system is evaluated. A workload should have the following characteristics:

- Measurable: A metric that can be quantified, such as throughput and response time.
- Reproducible: The same results can be reproduced when the same test is executed multiple times.
- Static: The same results can be achieved no matter for how long you execute the run.
- Representative: The workload realistically represents the stress to the system under normal operating considerations.

Workload should be discerned based on the specifications of the system. If you are developing a data driven system, where 90% of the requests require a database connection, this is a significantly different workload compared to a Web application that makes 90% JSP and servlet calls, with the remaining 10% being messages sent to a back end store. This requires close work between the application architect and the system architect, which are rarely the same person.

The projected load for a new system should be determined as early as possible. In the case of Web applications, it is often difficult to predict traffic flows or to anticipate user demand for the new application. In those cases, estimate using realistic models, and then review the data when the system is launched to adjust expectations.

14.2.4 Default messaging provider scalability

With the introduction of the messaging engine and the service integration bus, WebSphere now offers the ability to scale messaging resources more efficiently. Using a cluster as the service integration bus member, you can create a partitioned destination. This allow a single logical queue to be spread across multiple messaging engines. In this scenario, all messaging engines are active all the time. For *n* cluster members, the theory would be that each would receive an *nth* of the messages. This allows for greater scalability of messaging resources across the cell. One key factor to consider in this design is that message order is not preserved. That might or might not be significant, depending on the nature of the application.

14.2.5 Scalability checklist

When designing for scalability, several components are routinely found to be the source of bottlenecks. This section provides a quick list of items that should be checked multiple times during the implementation and load testing so that the system is scalable when it is launched.

- Physical bottleneck areas
  - Network Load Balancers
  - Firewalls
  - Application servers
  - Database servers
  - LTPA provider servers

- Logical bottleneck areas
  - JVM
    - Minimum heap size
    - Maximum heap size
  - Web Container
    - Maximum Persistent Requests
    - Timeout values
  - EJB Container
    - Inactive Pool Cleanup Interval
    - Cache size
  - Database connection pool
    - Maximum connections
    - Unused timeout
    - Purge policy
  - Database Server
    - Maximum Database Agents
    - Maximum Connected Applications
    - Query Heap Size
    - Sort Heap Size
    - Bufferpool size
    - Database memory heap
    - Application Control heap
    - Lock timeout
  - Directory Services
    - Database tuning
    - Authentication cache intervals

14.3 Workload management

Workload management is the concept of sharing requests across multiple instances of a resource. In WebSphere Application Server V6, this is embedded in the sharing of requests across one or more application servers, each running a copy of the Web application. In more complex topologies such as those described in Chapter 9, “Topologies” on page 245, workload management is embedded in load balancing technologies that can be used in front of Web servers or Lightweight Third Party Authentication Protocol (LTPA) providers.

Workload management techniques are implemented expressly for providing scalability and availability within a system. These techniques allow the system to serve more concurrent requests. Workload management allows for better use of resources by distributing load more evenly. Components that are over worked, and therefore, perhaps a potential bottleneck, can be routed around with workload management algorithms. The management techniques also provide higher resiliency by routing requests around failed components to duplicate copies of that resource.

In a WebSphere Application Server V6 environment, HTTP requests, servlet requests, and EJB requests can be workload managed. A new feature in WebSphere Application Server V6 is the WLM function of the service integration bus. This bus allows messaging requests to be sent to any existing server in the service integration bus and the request are routed to the cluster member that is running the messaging engine in that cluster.

14.3.1 Scheduling

Scheduling is a facility to queue and prioritize asynchronous requests in a system. WebSphere Application Server V6 uses scheduling for asynchronous beans that process user requests.

Asynchronous beans can inherit J2EE context information by enabling one or more J2EE service contexts on the work manager resource in the WebSphere administrative console or by setting the serviceNames attribute of the WorkManagerInfo configuration object. When a J2EE service context is enabled, it propagates the context from the scheduling thread to the target thread. If not enabled, the target thread does not inherit the context of the scheduling thread and a default context is applied. Any related J2EE context that is already present on the thread is suspended before any new J2EE context is applied.

The context information of each selected service is propagated to each work or alarm that is created using this work manager. Selecting services that are not needed can negatively impact performance.

14.3.2 Quality of Service

When working with any environment, performance planning should take into account Quality of Service (QoS) constraints. For workload management, this can mean that different types of requests that need to be processed at higher or lower priority.

Within the WebSphere Application Server V6 service integration bus, you have the ability to specify the reliability of message delivery, which impacts the workload management as it applies to messages and the messaging engine. You can even design the messaging engine to use aliases, which allows you to assign different QoS constraints to the same location, depending on how the location is accessed. The WebSphere Application Server V6 term for this type of message manipulation is *mediation* and is only available inside WebSphere Platform Messaging. When a message is sent to an external messaging provider, no additional mediation can be performed.

14.4 High availability

When designing and planning for a WebSphere Application Server environment, high availability should also be considered. High availability is the concept that describes an overall system resiliency. Because it is likely that the complete environment is made up of multiple systems, the goal is to make the system as available as possible. This can be done through minimizing any single points of failure (SPOF) throughout the system. If the SPOF cannot be eliminated, then the corresponding planning should be made to mitigate the impact of that potential failure.

14.4.1 Hardware availability

Hardware can and potentially will fail. Any mechanical component has an expected failure rate, and a projected useful life until failure. To help mitigate these types of failures, you can configure the equipment to have dual power supplies. With a dual power supply configuration, you can further mitigate power failures by plugging each power supply into separate circuits in the data center.

For servers, using multiple network interface cards in an adapter teaming configuration allows a server to bind one IP address to more than one adapters, and then provide fail over facilities for the adapter. This, of course, should be extended by plugging each adapter into separate switches as well, to mitigate the failure of a switch within the network infrastructure.

Hardware availability for servers at the disk level is also an important consideration. External disk drive arrays and hard disk drive racks can be deployed to provide redundant paths to the data, as well as make the disks available independent of server failure. When working with disks, consider the appropriate RAID levels for your disks.

RAID 0 is the simplest implementation. It involves striping the data across multiple disks. RAID 0, however, does not provide any degree of redundancy, and is used mostly from a performance standpoint. Most administrators will implement RAID 1 or disk mirroring as a minimum redundancy option. Be aware that different software components are impacted differently by the RAID levels. Databases, for example, that are write intensive, generally do not perform well with RAID 1, because the operating system is required to write the data twice to keep the disks in sync. RAID 5 is another common implementation. It uses bock striping of the disks, with a distributed parity. The availability of data on a disk is preserved by keeping the parity information separate from the data stripe. This environment can be good for written data, but is generally slower for reads. A more recent trend is to combine RAID levels, such as 0+1 or 1+0 (which is also referred to as RAID 10.) Using RAID 0+1 suggests striping the data across disks and then mirroring the stripe sets. RAID 1+0 mirrors the disks and then stripes across the disk mirrors. The difference is subtle, but significant. In a RAID 0+1 environment, if a disk fails in either stripe set, that set is lost, and the stripe set continues to operate without fault tolerance. This means that a failure of 2 disks in the correct positioning would result in a catastrophic loss of the data. In a RAID 1+0, The disks are mirrored first, and then the stripe sets are written to the mirrored copies. In this implementation, you can lose up to half of the disks without losing the data. Be aware that mirroring results in only half the capacity of the disks being available, and plan accordingly. Most major database implementations suggest using RAID 1+0 for optimum performance.

Network hardware availability can be addressed by most major vendors. There is now built in support for stateful failover of firewalls, trunking of switches, and failover for routers. These devices also support duplicate power supplies, multiple controllers, and management devices.

Duplicating hardware can have an impact on the cost of the solution. You have to evaluate this increment in the implementation cost against the cost of not having the application available.

14.4.2 Process availability

In WebSphere Application Server V6, the concept of a singleton process is introduced. While not a new concept, it is important to understand what this represents in the environment. A singleton process is an executing function that can exist in only one location at any given instance, or multiple instances of this function operate independently of one another. A messaging provider, for example, is considered a singleton process. In previous versions, the jmsserver was considered a singleton process, because multiple copies did not cooperate with one another, or only one process could be considered active.

In any system, there are likely to be singleton processes that are key components of the system functionality. WebSphere Application Server V6 has recognized this as a problem, and has taken steps to address these issues for singleton processes running in the WebSphere Application Server cell. The introduction of the service integration bus has provided the cell the ability to enforce high availability rules on processes that before were potential single points of failure.

For the most part, the service integration bus is used to provide high availability to the messaging system process. If the service integration bus member is created as a cluster instead of a single server, if the cluster member running the messaging engine fails, and the High Availability Manager is configured for the messaging engine, the service integration bus starts the messaging engine on another member in the cluster. To accomplish the fail over seamlessly, the queue information must be stored in some shared location, either using an external database, or a shared disk environment using the Cloudscape environment. For shared disks, concurrent access can be a concern. Cloudscape does not support multiple servers running the Cloudscape engine, so there would be no ability to have multiple servers communicating with the same shared file system. Most database engines have minimal support for distributed row locking. This means that it is generally difficult for more than one application to update a data element at the same time. Because the messaging engine is a singleton process, there are no concerns over concurrent access, or issues with distributed row locking requirements. Only one cluster member at any given time is executing the process, although any cluster member has the ability to spawn the process in case of failure of the active cluster member.

14.4.3 Data availability

In a WebSphere Application Server environment, there are many places where data availability is important. For database driven applications, the database availability is instrumental to the successful execution of the user request. For stateful applications, the user's session state is important to maintaining the user experience within the application. Stateful session EJB availability is similar in nature to the session state of a servlet request. Lastly, EJB persistence is critical in EJB driven applications to maintain a variety of user data. The majority of these requirements can be satisfied using facilities available in WebSphere Application Server V6.

Database server availability

A database server is for many systems the largest and most critical single point of failure in the environment. Depending on the nature of this data, there are many techniques that you can employ to provide availability for this data.

For read only data, multiple copies of the database can be placed behind a load balancing device, and the client then connect to the Virtual IP the load balancer advertises. This allows the application to connect to one of many copies of the data, and transparently fail over to the working copy of the resource.

If the data is mostly read-only, consider the possibility of replication facilities to keep multiple copies synchronized behind a Virtual IP. Most commercial database management systems offer some form of replication facility to keep copies of a database synchronized.

If the data is read/write and there is no prevalence of read-only access, then consider a hardware clustering solution for the database node. This requires external shared disk via SAN, NAS, or some other facility that can help to mitigate failure of a single node.

Session Data

For persistence of session data, WebSphere Application Server V6 supports memory to memory replication using JMS or persistence using an external database. The choice of which to use is really left to the user. External database persistence will survive node failures, and reboots to application server systems, but introduces a new single point of failure that must be mitigated using an external hardware clustering or high availability solution. Memory to memory replication using JMS can reduce the impact of failure, but the nature of the messaging is that these messages are not guaranteed, and as such, if a node were to fail, the messages held on that node would be lost.

Stateful Session EJB

Stateful Session EJB availability is handled using memory to memory replication, much like HTTP session state is maintained were is no external database facility. However, each application server definition has an EJB container.

Using the container properties, you can specify a replication domain for the EJB container, and enable the stateful session bean failover using memory to memory replication. When enabled, all stateful session beans in the container are able to failover to another instance of the bean and still maintain session state.

EJB persistence

When designing applications that use the EJB 2.0 and higher specification, the ability to persist these beans becomes available. If the beans participate in a clustered container, then the bean persistence is available for all members of the cluster. Using access intent policies, you have the ability to govern the data access for the persisted bean.

For the specific values that access intents can hold, check the WebSphere Application Server V6 Information Center at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

14.4.4 Clustering and failover

When discussing clustering, there are really two types of clustering: hardware and software. The section 14.4.1, "Hardware availability" on page 415 discussed techniques to make individual servers more highly available by mitigating the single point of failure within the server itself. Hardware clustering is the concept of creating highly available system processes across multiple servers. It is usually deployed in a manner such that only one of the servers is actively running the system resource. With software clustering, the idea is to create multiple copies of an application component and then have all of these copies available at once, both for availability and scalability. Software clustering as it pertains to WebSphere Application Server Network Deployment V6 is discussed in this section, although there are other software clustering facilities available.

Hardware clustering is achieved by using an external clustering software, such as IBM HACMP to create a cluster of servers. Each server is generally attached to a shared disk array through NAS, a Storage Area Network (SAN), or simply by chaining SCSI connections to an external disk array. Each system has the base software image installed. Depending on the configuration and licensing constraints, the software might be configured to run on each server in the cluster or configured to run from the shared disk so that only the active server has the software. The servers stay in constant communication with each other over several connections through the use of heartbeats. Multiple paths are configured for these heartbeats so that the loss of a switch or network interface does not necessarily cause a fail over. Too few paths can create problems with both servers believing they should be the active node. Too many paths can create issues in that the overhead associated with heartbeat management can degrade performance of the servers in the cluster.

In WebSphere Application Server Network Deployment V6, application servers can be clustered. This provides both Workload management, and high availability. Multiple copies of an application server can be defined in a horizontally cloned environment. Web containers or EJB containers can be clustered. If an application has both EJB and Web containers, you should strive to create two separate clusters, as this will increase the overall availability of the system, and potentially increase the performance. In a clustered environment, the plug-in module knows the location of all cluster members and routes requests to all of those members. In the case of a failure, the plug-in marks the cluster member as unavailable and does not send requests to that member for a fixed interval. The plug-in periodically retries the path to that cluster member and marks the member available again, after the path has been asserted. This retry interval is tunable through the administrative console.

14.5 Caching

Caching is a facility to offload some or all of the work to an external device or devices so that the application server is not required to do all of the work associated with a user request. There are caching options at many different layers in a complete system solution. This section provides an overview of the different possibilities for caching within a system. It does not attempt to provide all options, or specific details, because the implementation types of caching are highly varied.

14.5.1 Dynamic Caching

Dynamic caching refers to the methods employed by WebSphere Application Server to provide fragment caching or the reuse of components within the application server engine. Fragment caching means only the static portions of a servlet or JSP are cached, and is really a subset of dynamic caching as a whole. This caching is configured with two files at the WebSphere engine level. In WebSphere Application Server V6, these files are cachespec.xml and dynaedge-cfg.xml.

Another facet of caching available, beginning in WebSphere Application Server V5 fix pack 2, is the Edge Side Include (ESI) caching. ESI caching is an in-memory caching at the Web server plug-in. If dynamic caching is enabled at the servlet Web container level, then the plug-in uses the Edge Side Include caching. Additional headers are added to the request from the plug-in, called the Surrogate-Capabilities header, and the Application Server returns a Surrogate-Control header in the response. Then, depending on the rules specified for servlet caching, you can cache responses for JSP and servlets in the plug-in itself. For more specific details, you should review ESI caching in the InfoCenter at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

The cachespec.xml file allows you to configure caching at a per servlet level. The caching options include parameters the request might specify, and a timeout value to indicate the cache is no longer valid, and should be reloaded at the next request. Pay special attention to the servlet caching configuration, because you can create unexpected results by returning a cached servlet fragment that is stale.

The dynaedge-cfg.xml file is used by WebSphere Application Server V6 to configure the invalidation parameters used by external caching using an Edge Server that is WebSphere Application Server V6 aware. It includes information related to the URL, the domain, the passwords for invalidation, and the keystore for SSL communication.

14.5.2 Edge Caching

Edge caching embraces a variety of methods. Software components at the Web server, such as the Fast Response Cache Accelerator, can be deployed to provide caching of static content at the Web server itself. Reverse proxy engines, like WebSphere Edge Server can be used to cache content even closer to the customer. Lastly, there are numerous external caching providers that can provide content offloading at points significantly closer to the customer. WebSphere Application Server V6 can be used to configure and manage how these resources are accessed.

Fast Response Cache Accelerator

The first point to mention is that while WebSphere Application Server supports static file serving, the Web server is ultimately more efficient at this type of serving. By separating the two, the application will perform more efficiently. When the static content has been separated from the application content, IBM HTTP Server provides the Fast Response Cache Accelerator (FRCA) to speed up serving static content. The FRCA can be used to improve the performance of IBM HTTP Server when serving static content, such as images, HTML, and other text files that are not dynamic content. It is configured in the httpd.conf file. For further details about the configuration options, visit the IBM HTTP Server InfoCenter at:

http://www.ibm.com/software/webservers/httpservers/doc/v2047/manual/ibm/en_US/

This cache can even be configured to enable high speed caching of servlets and JavaServer pages files by using the Afpa adapter bean through the external caching configuration section of the administrative console.

WebSphere Edge Server

By using the WebSphere Edge Server Caching Proxy, you can intercept requests and then cache the results away from the Web servers or the application servers. This allows you to offload additional workload from the primary processing environment. Implementing this caching adds servers and cost to the solution, but can reflect a significant performance improvement, and a reduction in response time. This cache can be configured to offload some or all of the site. Using the WebSphere Application Server V6 administrative console, you can also control through WebSphere how the content is loaded and invalidated.

Hardware caching

There are many network equipment providers that sell hardware cache devices. These are very similar to a software cache, in the way they can be used to offload content. The main difference is that these appliances are usually not running full versions of an operating system, opting instead for a thinner operating system that is dedicated to the caching function. This can include custom file systems that are higher performance than some operating system file systems, and a significantly reduced instruction set. By placing dedicated appliances instead of software caching, you might be able to reduce total cost of ownership, because these appliances do not have to be managed as strictly as machines with full operating systems.

Caching services

There are a variety of providers that now sell caching as a service. This function can provide even higher performance gains, because these providers generally have equipment positioned at Internet peering points throughout the world. This means the user is not required to travel all the way through the Internet to get to the application serving network to return content. In other words, these providers bring the cached files physically as close as possible to the client.

Caching providers also can lower project costs in the long run in terms of less dedicated staff, equipment, and floor space to internalize the same function. The number of players in this market has exploded in the recent years. Making no preference or recommendation statements about any of the providers, some of the names involved are:

- Accelion
- Activate
- Akamai
- AT&T
- Digital Island
- EdgeStream
- Fireclick
- Limelight Networks
- Speedera

When selecting an external caching service, you should make sure they reach your target market. If your customer base is in Europe, and the service only has equipment in the United States, you might not see as much performance enhancement as you would like.

14.5.3 Data caching

Data caching can be a tricky proposition. The key is to minimize the back end Database calls while at the same time assuring the currency of the data. In most cases, the decision for data currency is a business decision. This is a decision that should not be taken lightly, especially if the data is provided by external sources. When configuring data caching, there are multiple methods. The first is to keep a local copy in a database within the same network realm as the application. The second is to cache data from a localized database in memory to minimize database reads. The next is to use EJB persistence to keep the data in the memory space of the running application.

In the case of some data, an external provider maintains the content. Making live calls to this data can prove to be a single point of failure, and a slower performer. However, there are no strict dependencies on the currency of the data, offloading this data or a subset to a local database can provide large performance, availability and scalability gains. The data can be refreshed periodically, preferably during off peak hours for the application. The provider must support this refresh process, which might not be appealing to them, depending on the nature of the relationship.

Caching data from a local database is a normal operation for a Database Administrator. There are facilities to minimize the number of times a query must perform a read from disk. Facilities like fetch ahead constructs attempt to anticipate that additional pages from that table are required and pre-loads those pages into memory pools. Buffer pools offer the ability to keep the data loaded into memory, assuming that it will be likely the same portion of the data will be requested again. Both of these constructs limits disk access, opting instead for reading the data from the memory, which is always much faster to read. These facilities necessarily assume the data is predominately read only. If the data has been written, then the copy in memory can be stale, depending on the write implementation of the database.

EJB persistence implies loading the data into an EJB after a call to the data provider. This is similar in nature to database caching, except that the caching takes place in the application space, not the database server memory. The EJB has an access intent, which indicates the rules used to determine the currency of the data in the bean. From a performance standpoint, avoiding a call to an external database in favor of a local bean would create significant gains.

14.6 WebSphere Application Server performance tools

When reviewing the application environment, you often need to delve deeper into the behavior of the application than what is presented at the operating system layer. This requires the use of specialized tools to capture this information. WebSphere Application Server introduced tools in Version 5 that allowed the administrator to gather information related to the performance of various components in the J2EE environment. WebSphere Application Server V6 has extended the power of these tools, and introduced new tools. In this section, we will discuss the enhanced Tivoli Performance Viewer and the Request Metrics now available for WebSphere transactions.

14.6.1 Performance Monitoring Infrastructure

The Tivoli Performance Viewer is a monitoring tool that is used to capture data presented through the WebSphere Application Server Performance Monitoring Infrastructure (PMI.) The PMI is a server side monitoring component. Configured through the administrative console, the PMI allows monitoring tools to peer inside the WebSphere environment to capture specific details about the application behavior. Using this interface, you are able to capture information about the following and more:

- Customer's application resources
  - Custom PMI
  - EJBs
  - Servlets/JSPs
  - Web Services
  - Any component written by the customer that runs in the environment
- WebSphere run-time resources
  - JVM memory
  - Thread pools
  - Database connection pools
  - Session Persistence data
- System resources
  - Processor usage
  - Total free memory
  - Components that are controlled outside the WebSphere environment but that are vital in healthy application state

Review details about the PMI data that is captured at the InfoCenter at:

`http://www.ibm.com/software/webservers/appserv/library.html`

14.6.2 Performance tool changes

Because of new facilities introduced in WebSphere Application Server V6, the PMI data was enhanced to include data on the service integration bus and detailed messaging as well as the High Availability Manager for tracking high availability components.

One new consideration for the PMI is the implementation of the J2EE 1.4 Performance Data Framework. This means that the old API has been deprecated, although it will continue to exist to allow for migration to the new standard.

The new PMI now offers custom PMI. This allows you to insert your own custom metrics and have them captured and available to the standard monitoring tools. The PMI also has removed the performance issues associated with previous versions. The new interface is to enable or disable a collection, instead of also setting the impact to the system.

The PMI grouping has changed. In previous versions, PMI metrics were grouped according to performance impact levels:

- Setting PMI monitoring to high enabled collection of PMI counters with performance impact level of high or less
- WebSphere Application Server V6 organizes PMI metrics into categories
  - Basic
    - J2EE components
    - CPU usage
    - HTTP session info
  - Extended
    - Basic
    - WLM
    - Dynamic cache
- Custom offers fine-grained control
- Sequential update
  - Causes all PMI counters to be updated sequentially
  - Enabling adds additional overhead

14.6.3 Monitoring impact on performance

Monitoring a system naturally changes the nature of the system. Introducing performance metrics consumes some resources for the application. In WebSphere Application Server V6, the development team sought to minimize the overall impact of the monitoring. Here are some considerations:

- All data collection impacts performance in some way
  - Basic setting has overhead of less than 2%
  - Basic setting is enabled be default
- Extended setting has overhead of less than 3%
- All setting has overhead of less than %
- Collection of Java Virtual Machine Profiler Interface (JVMPI) data adds additional overhead
  - Collecting just GC and Thread data adds less than 2%
  - Collecting all JVMPI data adds 20 to 25%

14.6.4 Tivoli Performance Viewer

The Tivoli Performance Viewer is the tool included with WebSphere Application Server V6 for measuring performance. Unlike its predecessor, however, this version is now integrated into the administrative console. Tivoli Performance Viewer is responsible for:

- Displays PMI data collected from local and remote application servers
  - Summary reports show key areas of contention
  - Graphical/tabular views of raw PMI data
  - Optionally save collected PMI data to logs
- Provides configuration advice via performance advisor section
  - Tuning advice formulated from gathered PMI and configuration data

You use Tivoli Performance Viewer to create summary reports. These reports let you monitor the server's real-time performance and health. Tivoli Performance Viewer allows you to work with the performance modules. These modules allow you to drill down on specific areas of interest, even including old logs. Use the log analysis tools to detect trends over time. Tivoli Performance Viewer can also save performance data for later analysis or problem determination.

14.6.5 WebSphere Performance Advisors

Gathering in the information made available through the PMI, the WebSphere Performance Advisors have the ability to make suggestions about the environment. The advisors are able to determine the current configuration for an application server, and trending the PMI data over time, make informed decisions about potential environmental changes that can enhance performance of the system. Advice is hard coded into the system, and is based on IBM best practices for tuning and performance. The Advisors do not implement any changes to the environment. Instead, they identify the problem, and allow the system administrator to make the decision whether or not to implement. As discussed in "Benchmarking" on page 208, you should test after any change is implemented from the Advisor suggestions. In WebSphere Application Server V6 there are two types of Advisor. The first is the runtime advisor, and the second is the advisor in Tivoli Performance Viewer. This section discusses those Advisors.

Runtime Performance Advisor

This advisor is configured through the administrative console. The Runtime Advisor writes to the SystemOut.log and to the console while in monitor mode.

The interface is configurable to determine how often data is gathered and advice is written. The Runtime Advisor offers advice on the following components:

- ORB service thread pools
- Web container thread pools
- Connection pool size
- Persisted session size and time
- Prepared statement cache size
- Session cache size

This is not the complete set of items monitored through the PMI. If you need to gather advice on items outside this list, you need to use the Tivoli Performance Viewer Advisor. The Runtime Advisor, by its nature, does not have the ability to play back captured data off-line.

Performance Advisor in Tivoli Performance Viewer

This advisor is slightly different than the Runtime. The Performance Advisor in Tivoli Performance Viewer is invoked only through the Tivoli Performance Viewer interface. It does still run on the application server you are monitoring, but the refresh intervals are based on selecting refresh through the console instead of fixed intervals. Also, the output is to the User Interface instead of to an application server output log. This advisor also captures data and gives advice on more components than the Runtime Viewer. Specifically, this Advisor can capture:

- ORB service thread pools
- Web container thread pools
- Connection pool size
- Persisted session size and time
- Prepared statement cache size
- Session cache size
- Dynamic cache size
- JVM heap size
- DB2 Performance Configuration

This Advisor is used more for calculation intensive operations. The metrics it monitors can become quite large and could potentially impact performance if analyzed through the runtime advisor. It is an on-demand tool, much more suited to problem analysis and avoidance.

14.6.6 WebSphere request metrics

Request metrics gather information about single transactions within an application. The metric tracks each step in a transaction and determines the process time for each of the major application components. In WebSphere Application Server V6, several components now support this transaction metric. These include:

- Web server plug-ins
- Web container
- EJB container
- JDBC calls
- Web Services container
- JMS messaging engine

The amount of time that is spent in each component is measured and aggregated to define the complete transaction time for that transaction. Both the individual component times and the overall transaction time can be very useful metrics when trying to gauge user experience on the site. The data allows for a hierarchical view for each individual transaction by response time. When debugging resources constraints, these metrics provide critical data at each component. The request metric can provide filtering mechanisms to monitor synthetic transactions or to track the performance of a specific transaction. By using artificial transactions, you can measure performance of the site end-to-end.

From a performance perspective, using transaction request metrics can aid in determining if an application is meeting Service Level Agreements (SLA) for the customer. The metrics can be used to alert when an SLA target is not met.

Request metrics help the administrators answer the following questions:

- What performance area should be focused on?
- Is there too much time being spent on any given area?
- How do I determine if response times for transactions are meeting their goals?
- How can I validate SLA agreements are being met?

Version 6 enhanced the request metrics from previous versions. Those familiar with the Application Response Measurement (ARM) standard should know that beginning in WebSphere Application Server V5.1, the environment was ARM 4.0 compliant. WebSphere Application Server V6 extended the attributes measured to include Web Services, JMS, and Asynchronous Beans.

14.6.7 Implementing request metrics

There are several methods for implementing request metrics. This section briefly discusses the methods that are currently available.

Request filtering

The most common method of implementing request metrics is to use request filtering. In this method, you use filters to limit the amount of transactions that are logged, capturing only those transactions you care to monitor. As an example, you could use an IP address filter to monitor synthetic transactions that always come from the same server. Some of the available filters are:

- HTTP requests: filtered by IP address, URI or both
- Enterprise Bean requests: filtered by method name
- JMS requests: filtered by parameters
- Web Services requests: filtered by parameters
- The performance impact is less than about 5 percent when all incoming transactions are being instrumented. This is not a significant amount, but should be factored in when implementing the metrics. The console path for implementation is through Monitoring and Tuning → Request Metrics.

Tracing

By setting the trace depth, you control not only the depth of information gathered through the metric, but also the overall performance hit on the system. The higher a tracing level, the greater the performance hit the system takes. There are several available trace levels:

- None: no data captured
- Hops: process boundaries (web server, servlet, EJB over RMI-IIOP)
- Performance Debug: Hops + 1 level of intra-process calls
- Debug: full capture (all cross-process/intra-process calls)

Tracing levels are set in the console through the same path, Monitoring and Tuning → Request Metrics.

Output for request metrics

The data captured by request metrics is placed in several levels, depending on the nature of the metric selected. For Web requests, the HTTP request is logged to the output file specified in the plugin-cfg.xml on the Web server. For Application Server layers, servlets, Web Services, EJB, JDBC, and JMS, the information is logged to the SystemOut.log for that application server. The data can also be output to an Application Response Measurement (ARM) agent and visualized using an ARM management software, such as IBM Tivoli Monitoring for Transaction Performance or IBM Enterprise Workload Management.

If you currently use a third-party tool that is ARM 4.0 compliant, the data can be read by that agent as well. The data can be directed to either the logs or the agent, or both at the same time. The best practice, however, is to not use metric logging while implementing the ARM agent monitoring, because the disk I/O can negatively impact performance.

Application Response Measurement

ARM is an Open Group standard that defines the specification and APIs for per-transaction performance monitoring. Request metrics can be configured to use ARM. In doing so, the request metrics use call across the ARM API to gather the data.

For more information about ARM, review:

http://www.opengroup.org/tech/management/arm/

WebSphere Request Metrics support the Open Group ARM 4.0 Standard, as well as the Tivoli ARM 2.0. The 4.0 standard is supported in all components. For the Tivoli standard, WebSphere Application Server V6 supports all components except Web server plug-ins. A correlator can be extracted from request metrics transactions. This correlator can be passed to sub-transactions taking place in non-WebSphere containers. This facility allows the complete transaction to be accurately timed in complex environments.

Additional resources

For additional information about ARM, and the WebSphere Request Metrics implementation, refer to the following links:

- To review the ARM 4.0 specification

  http://www.opengroup.org/management/arm.htm/
- Additional information about the ARM standard

  http://www.opengroup.org/pubs/catalog/c807.htm
- IBM Tivoli Monitoring for Transaction Performance

  http://www.ibm.com/software/tivoli/products/monitor-transaction/
- IBM Enterprise Workload Management

  http://www.ibm.com/servers/eserver/about/virtualization/suiteforservers/

15

Planning for messaging

This chapter discusses planning for a WebSphere Application Server V6 environment that uses messaging facilities. WebSphere Application Server V6 introduces significant changes to the messaging environment. Depending on your current messaging needs, you should read carefully the information that is provided in this chapter. This chapter contains the following sections:

- Messaging overview
- Messaging architecture
- Service integration bus resources
- JMS resources
- Security

15.1 Messaging overview

Messaging is a very popular facility for exchanging data between applications, and clients of very different types. It is also an excellent tool for communication between heterogeneous platforms. WebSphere Application Server V6 recognizes the power of messaging and has introduced a new messaging platform within the WebSphere environment.

This messaging environment is implemented through the use of a service integration bus which provides the framework needed to support service oriented architecture (SOA). This messaging environment is an important component of the enterprise-wide bus. This enterprise bus is commonly known as the Enterprise Service Bus or the Enterprise Nervous System.

Looking at the default messaging provider on WebSphere Application Server V6, the new facility offers significant power to applications. The environment offers integrated asynchronous capabilities for the WebSphere platform through a full JMS service provider that is fully JMS 1.1 compliant. The service integration bus offers an intelligent infrastructure for service oriented integration. It unifies SOA, messaging, message brokering, and publish/subscribe facilities. The default messaging provider is not meant to replace WebSphere MQ, but rather it is a mechanism to share and extend messaging family capabilities. The default messaging provider offers the following features:

- Multiple messaging patterns (APIs) and protocols for messaging and service oriented applications.
  - The default messaging provider is a fully J2EE 1.4 compliant JMS provider supporting the standard JMS APIs.
  - Implementation of relevant Web Services standards, including JAX-RPC APIs.
- Reliable message transport facilities.
- Intermediary logic called mediations to intelligently adapt message flow in the network.
- Clustering enablement for scalability and high availability (HA).
- Full Integration with WebSphere Application Server.
  - A full functioning JMS 1.1 provider
  - Full integration into server administration and runtime environment for:
    - Systems management
    - Reliability, availability, and serviceability (RAS)
    - Security
    - Performance Monitoring Infrastructure (PMI)
    - Java thread modeling and support

  – Uses the existing installation processes
  – Centralized management through the administrative console
  – An all Java implementation, with no external processes to manage

- Flexible Quality of Service (QoS) implementation.
- Connectivity to existing WebSphere MQ infrastructures.
- Uses any JDBC-compliant database for the message store.

The messaging in WebSphere Application Server V6 scalable and removes the single point of failure (SPOF) of the messaging server through HA facilities. The environment offers full connectivity to and co-existence with WebSphere MQ infrastructures as well as connectivity to other WebSphere Application Server V6 cells. Because the engine is now in-process, performance gains are also seen in the messaging environment. Lastly, the security is also now fully integrated.

The default messaging provider is more than just messaging. Web Services Integration through the service integration bus is an important feature of the new environment. The default messaging provider supports attachment of several Web Service applications, including:

- Requestors using the JAX-RPC API
- Providers running in WebSphere Application Server as stateless session beans and servlets using the JSR-109 standard
- Requestors or providers attaching via SOAP/HTTP or SOAP/JMS protocols

To access a Web service, you would view them as a destination in the default messaging provider. Using a Web Services Definition Language (WSDL) import, you can create the appropriate service destinations. Existing destinations, such as a JMS queue, can be exported to WSDL as a Web Service provider. The mediations that can be applied to destinations also can be used on Web Services destinations to influence the message content, routing, logging, or any other option available to destinations in the messaging engine.

15.1.1 Messaging considerations

Messaging can be a powerful technology for applications. It is important to remember, however, that not all environments need to use messaging. Just because WebSphere Application Server provides messaging and WebSphere Application Server V6 introduces a new messaging platform does not mean the technology is a panacea for applications. This section discusses in high-level statements considerations when choosing whether messaging is appropriate for your environment.

- Is your application complex or simple?

  An application that just uses servlets to provide Web presentation services for dynamic content, then it is unlikely it would use messaging to add any degree of value to the application. If, however, the application is data driven or has many tiers of access, then messaging can provide you a good solution.

- Does your application environment have multiple platforms?

  When dealing with heterogeneous systems, messaging can provide a common input and output mechanism that crosses platforms easily. If the environment is predominately homogenous, then messaging might not do any more than add a layer of complexity that would generally be undesirable.

- Does your application need to cross security realms?

  Some application environments require access to data that might not be in the same security realm as the application servers. This does not just mean a demilitarized zone (DMZ) environment, because some large enterprises use multiple security realms within an environment that would not at all be considered appropriate for a DMZ. Messaging provides Secure Sockets Layer (SSL) communication and cryptography to enhance the security of the system overall.

- Does your current environment use messaging?

  If your environment currently is a robust and stable WebSphere MQ environment, then the possibility is high that your applications will be more robust and stable if they implement messaging.

- What are your application resiliency requirements?

  Not all applications require high availability or high degrees of reliability. Introduce messaging into an environment where some back-end resource is a potential SPOF and you need to minimize the impact of that component being unavailable. Messaging itself can introduce SPOF, especially when integrating into existing messaging environments, so be careful as to how the choice is made.

- Are your developers comfortable with messaging technologies?

  Many developers have worked without using messaging for quite some time. They have found ways using session state, EJBs and other tricks to implement some of the same functionality without using messaging constructs directly. If the developers are not comfortable with using messaging, you can be turning on an environment that does nothing but consume resources that could be put to use elsewhere in the application environment.

There are many more questions that you can ask when considering messaging. The point really is that just because a guide discusses it or a software product offers it as a technology does not mean the application will make use of it. The best advice is to use the keep it simple and straightforward (KISS) method.

15.2 Messaging architecture

The default messaging provider is designed to be a robust and stable messaging platform. The messaging services are identical in a stand-alone or Network Deployment cell. This section discusses the high-level details of the architecture of the messaging environment created in WebSphere Application Server V6.

15.2.1 Basic messaging concepts

Messaging is a synchronous or asynchronous method of communicating between processes. It provides reliable, secured transport of requests between applications that might reside on the same server, different servers, or even different networks across a global application environment. The basic premise of messaging is an application produces a message which is placed on a destination or queue. The message is retrieved by a consumer, which then does additional processing. The end result can be that the producer receives some data back from the consumer or that the consumer does some processing task for the producer.

With the default messaging provider, the model is that a destination can be any number of points where communication rendezvous is managed. This can include JMS Queues, topics in a publish-subscribe environment, or Web Service endpoints. While the message is in a destination, the messaging engine can perform administration of the message, which can affect a variety of items that are related to the message.

15.2.2 Service integration bus

The service integration bus is a communication infrastructure that provides service integration through synchronous and asynchronous messaging. When a service integration bus is used for JMS applications, it is generally referred to as a messaging bus, and the terms at that point can be used interchangeably. In WebSphere Application Server V6, you can configure the service integration bus in a variety of ways. The most common implementation is to have one service integration bus in a stand-alone single server. It is, however, possible to have multiple inter-connected buses in a WebSphere Application Server Network Deployment V6 cell or even in a stand-alone or single server.

A bus is a cell wide messaging resource. A WebSphere cell can contain any number of buses. Each bus must contain one or more bus members. The exact decision as to how many buses and bus members you have depends on your messaging needs and cell topology. Some systems might require only a single bus with a single bus member, where other systems might require or benefit from a more complex messaging topology.

Some benefits of multiple messaging engine busses include:

- Improved scalability by spreading messaging workload across multiple servers.
- Performance enhancements from situating messaging destinations close to message producing or consuming applications.
- Improved availability by removing SPOF.

The ability to have multiple buses within a single cell can be beneficial. It enables completely separate messaging systems to be used for different purposes, such as having test and production systems where destination names need to be the same, or enables the separation of messaging applications which have no need to communicate with each other.

Keep in mind that the name of a bus is a fairly important choice as the name cannot be changed after the bus is created. The name should be meaningful and must be unique within the set of buses that can (in the future) be interconnected.

For example, if you have a bus named *myBus* and at some point in the future you need to create a link from it to another bus also called *myBus*, then there is no way to distinguish between the two buses for message routing, and a link cannot be created. Other buses that you can make links to include service integration buses in other WebSphere cells and WebSphere MQ queue managers.

In WebSphere Application Server a service integration bus consists of several resource types. For more details about these resources, see 15.3, “Service integration bus resources” on page 444.

15.2.3 Messaging store

The messaging engine uses a data store for storing messages, transaction states, and delivery records. The messaging engine requires persistent backing data store. In WebSphere Application Server V6, this is a JDBC-compliant database. If your environment has multiple messaging engines, they are able to share a single database, but each engine has its own schema within the database. This means that two messaging engines that are active cannot share the same set of tables.

The messaging engine data store is important in several aspects. If designing your system for HA, where the bus member is a cluster instead of a stand-alone server, this store must be accessible by all cluster members in the cluster. In WebSphere Application Server V6, the default database is a Cloudscape database. This database, however, is not by default accessible by more than one server. Because of the requirement for accessibility from all cluster members, you have to either reconfigure Cloudscape to be a networked datasource or move it to an external database provider such as IBM DB2. Refer to *WebSphere Application Server V6 - System Management and Configuration Handbook*, SG24-6451 for specific implementation details. Note the additional licensing requirements for an external data source if you do not already have a licensed database manager in the environment.

15.2.4 Linking destinations to bus members

All bus destinations are associated with one or more bus members, which then are associated with a corresponding messaging engine. This allows the administrator to control which database is used for persistence. For the most part, a destination is associated with one messaging engine. Destinations are covered in more detail in 15.3.5, “Destinations” on page 446.

Basically, a queue is used for point-to-point messaging, and a topic space is used for publish-subscribe messaging. Every messaging engine in the service integration bus is a publication point where messages are held. When a client is doing point-to-point messaging with a destination defined on the local bus, it is able to both produce and consume messages. When a client is doing point-to-point messaging with a destination defined on a foreign bus it is only possible for the client to produce messages to that destination.

For users familiar with WebSphere MQ, this is similar to the restriction placed on clients that they must connect directly to the queue manager on which the queue is defined in order to be able to consume messages from that destination. If a client needs to consume messages from a point-to-point destination that exists on a foreign bus then the client must connect directly to that bus to consume the messages.

If it is not possible for the client to connect to the foreign bus directly then it is the responsibility of the foreign bus to forward the messages to a destination on the local bus from which the client can consume.

With publish/subscribe messages it is possible to configure a link such that a message which is published on either bus will be published on both. Note that, as with point-to-point destinations, it is not possible for clients to subscribe to a topic in a foreign bus. The link re-publishes messages on the other side of the link and the message consumers consume the messages from a topic on the local bus.

15.2.5 Connecting to a destination on a foreign bus

You can configure a connection to a destination on a foreign bus in the service integration bus layer and in the JMS layer. If you configure the connection in the service integration bus layer, then the JMS layer is unaware that the destination is on a foreign bus. If you configure the connection in the JMS layer, it is possible for the service integration layer to configure a foreign destination that overrides the defaults that are set on the foreign bus link.

Restriction: If an application is connecting to a queue that is actually on a foreign bus, it is only possible to produce messages for it. The applications cannot consume messages from it.

Service integration bus configuration

By creating an alias destination which defines both a bus name and a destination name on your service integration bus, clients to this service integration bus can send messages to the destination on the foreign bus without being aware that the destination is actually on a foreign bus.

JMS configuration

When defining a default messaging provider JMS queue resource, it is possible to specify a bus name in addition to a queue name. When the JMS client makes a connection with a connection factory, the bus name defined in the connection factory defines which bus the client will be connected to (the local bus). If the JMS client then uses a JMS queue that has a bus name in it, then the service integration bus to which the client is connected attempts to route any messages sent to the named destination on the named (foreign) bus.

When configuring in the JMS layer, it is possible to use a foreign destination to allow the service integration bus layer to set up default values for particular foreign destinations that override the destination defaults set on the foreign bus.

15.2.6 Service integration bus link

A service integration bus link allows your service integration bus to exchange messages with another service integration bus. The foreign service integration bus does not need to be in the same WebSphere cell as the local bus.

Topic space mappings

It is possible to link a topic space in the local bus to a topic space in the foreign bus to allow messages published on the foreign topic space to also be published on the local one. This is done in the link routing properties of the foreign bus definition.

Should you wish to prevent a foreign bus to which you have a link from receiving messages published on your own topic spaces, then configure the link on your side with a user ID that does not have permission to subscribe to the topic spaces you wish to secure.

Note: If no topic space mappings are defined, then there is no exchange of message publications across the link.

15.2.7 WebSphere MQ link

A WebSphere MQ link allows your service integration bus to exchange messages with a WebSphere MQ Queue Manager. The link does this by appearing to be another queue manager to which your MQ queue manager can be configured to talk. The MQ Link defines a queue manager name and sender and receiver channels. The receiver channel defaults to listening on port 5559 which enables an MQ Link to coexist on the same host as an MQ Queue Manager. When sending and receiving messages to and from MQ , the MQ link performs format conversions on the messages to ensure that they are understood by the message receiver.

MQ publish/subscribe broker profile

It is possible to set up an MQ publish/subscribe broker profile. This allows you to define topic mappings between service integration bus topic spaces and MQ Broker topics. This allows subscribers in the service integration bus to receive messages published in MQ broker, and subscribers to an MQ broker topic to receive messages that are published on a topic in the service integration bus.

The broker profile is configured on the WebSphere MQ link.

Accessing queues in a WebSphere MQ network

When sending a message to a queue that exists in WebSphere MQ from a service integration bus there are several naming considerations that you must take into account. If the queue you wish to access is defined on a queue manager that has the same name as the name of the foreign bus defined on the service integration bus, then the name of the queue you access need only be the name of the queue.

If the queue manager has a different name to that of the foreign bus used to access it then the queue manager name must be specified as a suffix to the queue name, for example *myQueue@myQueueManager*. This naming allows the message to be appropriately routed by WebSphere MQ when the message has left the service integration bus. If no queue manager name is applied as a suffix, then the foreign bus name is added as the queue manager name by default.

Note: It is possible to address a queue that is defined on a queue manager other than the one that is participating in the MQ link while using a direct, WebSphere MQ link foreign bus. However, it is not possible to address a queue that exists on a foreign bus to a direct, service integration bus link foreign bus. You must instead define an indirect foreign bus that targets the ultimate foreign bus where the destination is defined.

For further details about WebSphere MQ link, refer to *WebSphere Application Server V6 - System Management and Configuration Handbook*, SG24-6451.

15.2.8 Indirect link

An indirect link specifies a foreign bus that is not accessed directly but is accessed through another bus. For example, a WebSphere MQ queue manager that is accessed by going through a service integration bus or a service integration bus that is being accessed through another service integration bus are both indirect links.

15.2.9 WebSphere MQ client link

A WebSphere MQ client link enables a messaging engine to act as a WebSphere Application Server V5 or later embedded JMS Server. This function is provided as an aid to migration of WebSphere Application Server V5 or later to V6 and should not be used for any other purpose. It enables any applications that are installed and configured on WebSphere Application Server V5 or later, which use V5 or later JMS resources, to continue to function normally after the V5 or later JMS server has been migrated to WebSphere Application Server V6.

The process of migrating a WebSphere Application Server V5 or later node that contains an embedded JMS server removes that JMS server and creates a service integration bus with a WebSphere MQ client link. Queues that were defined previously on the WebSphere Application Server V5 or later embedded JMS server are created automatically on the service integration bus.

For further information about migrating the WebSphere Application Server V5 embedded messaging, refer to the WebSphere Application Server Information Center at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

Important: It is recommended that you replace all WebSphere Application Server V5 or later JMS resources with WebSphere Application Server V6 default messaging provider JMS resources as soon as possible. After you have changed resources, you can delete the WebSphere MQ client link, because all applications will be using the WebSphere Application Server V6 default messaging provider directly.

Note: You should not attempt create a WebSphere MQ client link manually. Use the one created automatically for you by the migration process.

15.2.10 Interoperability and WebSphere MQ integration

The default messaging provider offers full interoperability with other service integration buses in the same or different cells. The messaging engine also supports interoperability with WebSphere Application Server V5 Embedded JMS Servers. Existing WebSphere Application Server V5 embedded JMS clients can connect to V6 destinations. A WebSphere Application Server V6 JMS application can connect to an embedded JMS provider hosted in a V5 server. An MQ client link can be created to support any old WebSphere Application Server V5 clients to talk to WebSphere Application Server V6 messaging engines. Note that, however, there is no support for connecting a WebSphere Application Server V5 Embedded JMS Server to a WebSphere Application Server V6 service integration bus.

There is a strong relationship between default messaging provider and WebSphere MQ. The default messaging provider is not intended to replace the existing WebSphere MQ. The two products can coexist on the same machine without incident. They do not share any common modules or configuration data, so you do not have to be concerned about the configuration of one changing the configuration of the other. Connectivity between a messaging engine and an MQ queue manager is established by defining an MQ link. WebSphere MQ sees a WebSphere Application Server V6 messaging engine as another queue manager.

The link converts between the formats and protocols used by WebSphere MQ and default messaging provider. WebSphere MQ applications can send messages to queues hosted on default messaging provider and default messaging provider applications can send messages to WebSphere MQ queues. However, this does not imply that there is 100% interoperability between the two products. An MQ queue manager cannot attach to the default messaging provider bus using any communications protocol other than TCP/IP. Also, a default messaging provider cannot participate in a WebSphere MQ cluster.

Note: A messaging engine that supports the MQ Link cannot be a cluster member. It must be a stand-alone server because of the nature of channel communication in WebSphere MQ.

15.3 Service integration bus resources

There are five distinct resources that make up the service integration bus environment. Applications that use the service integration bus should pay close attention to how these resources are configured in the environment.

15.3.1 Bus

A bus, or service integration bus, is a component of WebSphere Application Server V6 that supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

15.3.2 Bus members

Bus members of a service integration bus are application servers or clusters on which the messaging engines are defined. When a new bus member is defined, a messaging engine is created automatically on the application server or cluster. It is important to note that there is only one messaging engine created even if the bus member is a cluster. If you want to have multiple messaging engines to provide scalability, you have to define additional messaging engines on the cluster. When a bus member is removed, the messaging engines that are associated with this member are also removed. In a WebSphere Application Server Network Deployment V6 cell, you can have multiple buses. Each bus can have servers or clusters as bus members.

15.3.3 Messaging engine

Messaging engines run inside the bus member application server and manage messaging resources. The messaging engine maintains a data store in a JDBC-compliant database that is used for messaging persistence. Queue-like destinations are associated with one or more messaging engines. The destinations allow the administrator to control which database or schema within a database is used for persistence. Messaging engines also provide a connection point for clients to put messages to or get messages from destinations.

15.3.4 Data store

A data store is the component of a messaging engine that handles persistence of messaging state in a database. Every messaging engine has a data store. A data store is a set of database tables that are exclusively accessed by a messaging engine and used to harden (write to disk) messages that need to be persisted. These database tables are accessed using a JDBC data source, of which the JNDI name must be configured in the messaging engine. A messaging engine prevents other messaging engines from accessing the same data by storing a unique identifier in the database.

Tip: This restriction might cause occasional difficulties in test environments where a messaging engine can be created and deleted several times.

Databases, user names, and schema names

Every messaging engine must have exclusive access to its own schema in a database. There are a variety of databases that are supported, including Cloudscape, Network Cloudscape, DB2, Oracle, MS SQL Server, Sybase, and Informix. Several of these databases have significantly different behaviors for schemas which are detailed in the following sections.

Note: An alternative to configuring schemas and user IDs is for every messaging engine to have its own database.

Important: If the data store for a messaging engine is configured to create tables, then the user ID used to access the database must have sufficient authority to create and drop tables. Check this with your database administrator.

If the user ID used to access the database does not have authority to create tables, then the database administrator must create the tables before starting the messaging engine. For further information about creating data store tables, refer to the WebSphere Application Server Information Center at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

Embedded Cloudscape (the default data store)

The service integration bus can create a data store for a messaging engine using a Cloudscape database that is embedded in the application server. This Cloudscape database enables you to get started quickly with the service integration bus. A JDBC data source to access the Cloudscape database is created at server scope on the server that has been added to the bus.

The default Cloudscape data store is not supported for cluster bus members. This is because multiple processes cannot access the database simultaneously which can happen during failover in a cluster. The Cloudscape database is given the same name as the messaging engine, which is unique. The embedded Cloudscape does not require an authentication alias to be set up on the data store.

Networked Cloudscape and DB2

Both of these databases allow the same user ID to access a single database concurrently with different schema names to access different tables. Thus, you can configure multiple messaging engines to access the same DB2 database, using the same user ID, but with its own unique schema name. Accessing the database with different user IDs automatically makes the actual schema accessed different.

When using Networked Cloudscape without security enabled, an authentication alias is required on the data store. However, the user name and password needs to contain only non-null values.

Restriction: If you are using DB2 for z/OS, an exception will be thrown at runtime if you leave **Create tables** selected on the data store for a messaging engine. Deselect **Create tables** and refer to the WebSphere Application Server Information Center for details about how to create the required tables for DB2 on z/OS.

Other Databases

Some databases do not make use of the specified schema name when a client connects. Thus, if you want to have multiple messaging engines accessing the same database, you need to assign each messaging engine a unique user ID that has permission to access the database. Refer to the documentation for your database for more information.

15.3.5 Destinations

A destination is a virtual place within a service integration bus. Applications, whether consumers or producers of messages, attach to a destination to exchange messages. Destinations within the service integration bus can be permanent or temporary. Permanent are created by the administrator and can only be deleted by the administrator. Temporary destinations can be created through API specific destinations. The most common use of a temporary destination is to specify a JMSReplyTo destination within a message.

There are several types of destinations that can be configured:

- Queue: Much like a queue in WebSphere MQ, used for point-to-point messaging.
- Topicspace: Used for publish/subscribe messaging such as that used in the WebSphere MQ Event Broker.
- Alias: A destination that is an alias for another target destination. Using an alias allows you to specify different rules and mediations for those messages.
- Foreign: A destination that identifies a destination on another service integration bus.
- Exception: A destination that is used to handle messages that cannot be sent to the intended service integration bus destination. Analogous to a dead letter queue.

From a logical perspective, destinations are created on the service integration bus and are hosted on a messaging engine. You are required to determine which messaging engine hosts which destinations in the service integration bus. Destinations also support QoS settings for reliability. QoS is a trade-off and should be examined closely. When increasing reliability, you decrease the performance of the destination. Destination QoS is generally set by the administrator during the creation. It is also possible for producers and consumers to specify QoS under application control using a combination of API calls and the connection factory used. The connection factory defines the persistence mapping. The following list describes the levels of QoS from lowest to highest reliability. This order also is from highest performance to lowest performance by the nature of the reliability.

- BEST_EFFORT_NONPERSISTENT

  As the name implies, these are non-persistent messages. They are never written to disk, and if the memory cache is overrun, the oldest messages are discarded. There is no acknowledgement that the message was received.
- EXPRESS_NONPERSISTENT

  Messages using this QoS are written asynchronously to persistent storage if the memory cache is overrun but are not kept between server restarts. There is no acknowledgement from the messaging engine that the message was received.
- RELIABLE_NONPERSISTENT

  These messages are the same as EXPRESS_NONPERSISTENT except that the client will now wait for a low level acknowledgement message. This will hold the application waiting for a response from the messaging destination.

- RELIABLE_PERSISTENT

  This class of messages is written asynchronously to persistent storage during normal processing and do survive server restarts. Because of the asynchronous nature of the writes, messages held in memory cache during a server failure are lost. Acknowledgement messages are sent to the client for this QoS on the destination.

- ASSURED_PERSISTENT

  This is the highest degree of reliability. Messages sent to destinations with this QoS are able to take advantage of assured delivery. The messages are written synchronously to persistent storage, so there is less likely a problem with server failure and messages held in memory cache. Acknowledgement messages are sent to the client for this QoS on the destination.

15.3.6 Mediations

Mediation in WebSphere Application Server V6 is defined as the ability to manipulate a message as it traverses the messaging bus. Mediation takes place at a destination and is attached administratively to that destination. When using mediation, one of the following occurs:

- Messages are transformed, where one or more attributes in the message header are changed.
- Messages are rerouted, where the destination of the message is changed.
- Messages are copied and routed to additional destinations.
- Interaction with non-messaging resource managers such as a database is allowed.

You can build your own mediation rules using supplied mediation subcomponents, known as Mediation Beans, that shipped with default messaging provider, or Mediation Beans that are supplied by IBM or a third-party not bundled in default messaging provider.

15.4 JMS resources

JMS resources are analogous to service integration bus resources, but they are not the same. Pure JMS communication uses connection factories, queues, and activation specifications. If you intend to use a WebSphere Application Server V6 messaging engine for use as JMS resources or to have those resources accessed by WebSphere Application Server V5 applications, then there are additional configuration steps to complete. This section covers the considerations for managing JMS resources in the WebSphere Application Server V6 environment.

15.4.1 Components used in a JMS configuration

This section discusses exactly what components are used when using JMS to access a messaging system. This section does not cover architectures for using JMS within applications.

When an application is written to access an asynchronous messaging system through JMS, that access is provided in one of two ways:

- Generic JMS usage.

  Using the JMS API to send and receive messages. The application relies on WebSphere Application Server to locate the relevant messaging system and to handle the connections to it. What the application does with sending and receiving the messages is entirely down to the application developer within the bounds of the JMS API.

- JMS and message-driven beans (MDBs).

  The application relies on WebSphere Application Server to monitor the underlying messaging system and pass any new messages to an instance of a purpose built MDB that will handle the message.

The more locations in an application that use JMS, the harder it is to configure optimally the JMS components. There are many relationships and dependencies between the components, some of which have already been discussed. When configuring a component in the JMS infrastructure, it is essential that you understand what resources are needed to make it work. When changing certain parameters, the requirements that the change places on the remainder of the components is not always obvious.

15.4.2 Generic JMS usage

Putting a message onto a message queue using JMS requires a number of components, including:

- JNDI Name service

  Described in detail in *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392

- Queue connection factory (QCF)

  Encapsulates the settings necessary to connect to a queue-based messaging system.

- Queue destination

  A reference to the point-to-point messaging queue.

- Queue manager

  A queue manager is a WebSphere MQ term. It refers to the component that is responsible for looking after a number of queues within one install of WebSphere MQ server.

- A message queue

  The actual queue where messages are held until removed by an application.

The JNDI namespace is used to house a link to the Java objects that will perform communication with the messaging system. The application performs a JNDI lookup for the queue connection factory using the resource reference. The resource reference is matched to the JNDI entry that contains the queue connection factory. The QCF object is returned. It is used to get a connection to the queue manager that is responsible for the target queue. A second JNDI lookup is performed and resolved to the correct JNDI entry. This time, it is for the queue destination, this will be used to locate the required queue from the queue manager. To be able to communicate with the queue in the messaging system, the application must first create a QueueConnection from the QCF object.

WebSphere Application Server maintains a pool of QueueConnection objects per QCF for use by any application in the application server. This request to create a new QueueConnection obtains an existing QueueConnection from this pool if there are any available. Otherwise, one is created and added to the pool.

When using the Embedded JMS server or WebSphere MQ as the underlying messaging system, the creation of the QueueConnection attempts a physical connection with the queue manager to verify that it is available.

The next step is to create a QueueSession object using the QueueConnection method createQueueSession. This is the stage where a physical connection is established to the queue manager which eventually is used to transmit the message to the queue. WebSphere Application Server maintains a pool of QueueSession objects for each established QueueConnection on the QCF. This request to create a new QueueSession obtains an existing QueueSession from this pool if there are any available. Otherwise, one is created and added to the pool.

With a connection now established to the messaging system through the QueueSession object, the application now specifies what sort of action is going to be performed. In this case, it is to send a message. The queue definition that was taken from the JNDI name service is passed to the QueueSession object.

This action tells the queue manager the specific queue on to which the message is placed. The message is constructed and sent to the message queue.

This is just a basic example of how JMS might be used within an application. However, it demonstrates the number of components that are used just to send a message. The configuration of these components can be crucial in making sure that the application performs fast enough for requirements.

15.4.3 JMS connection factories

In JMS 1.1, a new connection factory is introduced. This is the new unified connection factory. This unified factory allows for JMS connections for both point-to-point and publish-subscribe messaging styles. You can define the QoS for non-persistent message reliability just like you could for service integration bus destinations. You can also specify remote target types, if sending to external messaging resources. If you prefer, you can still create separate queue connection factory and topic connection factory definitions. This option might be preferred if you want to use different levels of reliability.

15.4.4 Service integration bus and JMS destinations

When an application needs to access a JMS queue or topic, there are two different administratively created objects. You would create the JMS destination and the service integration bus destination. In standard JMS, a JMS destination is simply a queue or a topic. In a service integration bus, these same resources are a service integration bus queue or a TopicSpace. The JMS destinations act as a proxy to the service integration bus destinations. JMS code communicates to the JMS Destination using regular JNDI namespace lookups. The destination is ultimately the service integration bus resource but should be invisible to the user. When working with topics (in a publish-subscribe message), the service integration bus treats a message sent to the JMS topic as a message sent to the associated TopicSpace. The actual topic becomes a message selector for the appropriate TopicSpace.

15.4.5 Service integration bus and MDBs

MDBs that are attached to destinations in the service integration bus are attached by means of the service integration bus JMS resource adapter, an activation specification, and a JMS destination. The resource adapter is responsible for connecting to the service integration bus and delivering messages to the MDB.

Note: It is recommended for performance reasons that MDBs always be installed on a server that has an active local messaging engine. It is also recommended that a MDB queue is configured to attach to has a queue point on the messaging engine local to the MDB.

15.4.6 MDBs connecting to the bus

The resource adapter always attempts to connect an MDB to a messaging engine in the same server if one is defined there. If there is no messaging engine in the same server, then a messaging engine is selected from the bus using the standard connection selection process.

There are three scenarios, discussed in the following sections, where an MDB is started but is not connected to the destination to which it is configured to listen. The resource adapter allows the MDB application to start under these circumstances and attempts to connect the MDB to its configured destination as soon as possible.

Local messaging engine defined but unavailable

If a messaging engine is defined locally but is unavailable when the MDB application starts, the application starts successfully and the resource adapter connects it to the messaging engine when it activates. Situations when this scenario happens include:

- If the messaging engine has not started by the time the MDB application is started.
- The MDB is installed on a cluster bus member which has been configured for high availability and is on a server other than the one with the active messaging engine.

When an MDB application is started but the locally defined messaging engine is unavailable, a warning message appears in SystemOut.log. If the messaging engine activates, then the MDB should connect correctly to the destination and report the connection in the same log.

Note: Messaging engines are frequently the last component of an application server to complete their startup, often even after the server is operational. So, it is not unusual for MDB applications to cause warning messages to be logged.

Remote destination unavailable

If there is an active locally defined messaging engine but the MDB is configured to listen to a queue that is currently unavailable (for example if the messaging engine that hosts those queues queue point is not active), then the MDB warns of the failure. The resource adapter tries to connect the MDB to the configured destination every 30 seconds until it succeeds. Each failure to connect results in a failure message.

Remote messaging engine unavailable

If there is no locally defined messaging engine, then a messaging engine is selected from the bus to connect to. If there are no currently available messaging engines in the bus, then the resource adapter allows the MDB application to start and attempts to connect the MDB to a messaging engine every 30 seconds. The MDB logs each attempt until the failure is resolved. The MDB is not delivered any messages until the resource adapter has been able to start a connection to an active messaging engine.

15.4.7 MDBs and clusters

The behavior of MDB installed on clusters that use the service integration bus is directly related to the service integration bus configuration. For more detailed descriptions of messaging configuration for clusters, refer to *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392.

Clusters that are not part of a bus

When a MDB is installed on a cluster that is not part of a bus, the MDB on each server connects independently to the bus to consume messages.

Note: You should not configure an MDB on a cluster with no local messaging engines that is listening to a partitioned queue in another cluster. There is no guarantee that every partition of that queue has at least one MDB listening to it.

Clusters configured for highly available messaging

When a cluster is configured for highly available messaging, a messaging engine will be active on one of the servers in the cluster. An MDB application installed on that cluster starts on all servers in the cluster, but only the MDB on the server with the active messaging engine on it receives messages. Should the active messaging engine fail or the server it is active on fail or be stopped, then the messaging engine starts up on another server in the cluster. The MDB on that server is connected to the messaging engine and start receiving messages.

In this scenario, the service integration bus has been configured to have one active messaging engine in the cluster and effectively the MDB mirrors that configuration.

Clusters configured for messaging workload management

When a cluster is configured for messaging workload management, a messaging engine most likely is active on each server in the cluster. If the MDB that is installed on the cluster is listening to a topic, then each message published on the topic is received once on each server with an active messaging engine. Should more than one messaging engine be active on a server, a topic message published still is received once by the MDB on that server.

If the MDB installed on the cluster is listening to a queue that is partitioned on the cluster, then the MDB is attached to each partition that is active on the server. Should more than one messaging engine be active on a server, then the MDB receives messages from each messaging engines partition of the queue.

If the MDB installed on the cluster is listening to a queue that has its queue point on a messaging engine outside of the cluster, then the MDB on each server is attached to the queue. If more than one messaging engine is active on a server, then the MDB does not consume a greater portion of the messages than an MDB on a server with only one messaging engine active on it.

15.4.8 MDB and activation specifications

MDB in the EJB 2.1 specification are executed inside an application server. Their primary function is to listen to queues and topics in a JMS environment. WebSphere Application Server V5 used listener ports to connect the MDB to a destination. New to WebSphere Application Server V6, the service integration bus uses a JCA connector to connect the MDB to the destination. Thus, you must define an activation specification that can be bound into the JNDI name space, so that when the MDB is started, the ActivationSpec connects it to the service integration bus destination. The ActivationSpec is created using the JNDI name of a JMS destination, which is a proxy for an service integration bus destination.

Existing EJB 2.0 MDBs can be deployed against a listener port, configured in an identical manner as is done in WebSphere Application Server V5. Alternatively, the MDB can also use the activation specification that is available in the default messaging provider. The best practice, however, is to deploy a new MDB using the EJB 2.1 specification and to migrate the old MDB to the new specification. You should plan this migration well as part of the implementation.

15.4.9 Component relationships when using MDBs

All the JMS components are linked together to allow an MDB to work when receiving point-to-point messages via a QCF. The relationships would be the same if it was for a topic connection factory (TCF). You can use these relationship to map the impact of making changes to the configuration.

These relationships become important when you make configuration changes, because each change could effect other components.

Table 15-1 shows some examples of using component relationships.

Table 15-1 Example linked relationships for JMS Components

Scenario	Requirements
Increase the number of queue sessions by one	Because there is only a single one-to-one relationship for a queue session, the only requirement to increase the number of queue sessions is to make sure that there are enough queue manager connections available. Remember, however, that by increasing the number of queue sessions in the session pool by one, it affects all the session pools for all the queue connections established from that QCF. So, it will in fact allow queue connections times one sessions, not just one more session. The setting for the maximum sessions in the session pool is determined by the queue connection that needs the most sessions.
Increase the listener port sessions by one.	Increasing the message listener threads has no impact on any other component, except that there needs to be enough message listener threads for all listener port's sessions.
Add a new listener port with one listener port session.	To do this, there needs to be enough message listener threads, queue connections, and queue sessions available. Increasing the number of queue sessions is not necessary, because this new queue connection has its own pool of queue sessions, which has at least one in the pool.

15.5 Component relationships using generic JMS

Component relationships are not as complex when using JMS to send and receive messages within your application code. This is much more dependent on how you code your application. As discussed in 7.2.3, "Categorizing your workload" on page 194, generic JMS calls means using QueueConnections and QueueSessions, whether it is work with publish/subscribe or point-to-point.

As mentioned earlier, QueueConnection objects are thread safe so they can be cached and then used across multiple threads in the application. However, QueueSessions are not thread safe, so each thread needs its own QueueSession.

To understand how the JMS components are related requires an understanding of how the application is using those components. Unlike using the message listener service, the application design or development defines how these components are used and what the relationship is between the components.

For a QCF or TCF, configuring the number of QueueConnections in the connection pool and QueueSessions in the session pool ultimately relies on knowing the way in which the code is written. Caching QueueConnections means that more QueueSessions are needed in the session pool.

- Example 1

  There are 30 application threads that all need to place a message on a queue through the same QCF simultaneously. One QueueConnection is obtained and shared across the threads. Thus, 30 QueueSessions are needed from that one QueueConnection. The connection pool maximum only needs to be one in this example, and the session pool maximum size needs to be 30.

  Using a new QueueConnection for each thread means that there is probably a one-to-one relationship between the QueueConnection and QueueSession that is used on that thread. So, there needs to be more QueueConnections in the connection pool and less QueueSessions.

- Example 2

  There are 30 application threads all needing to place a message on a Queue through the same QCF simultaneously. Each thread obtains a QueueConnection and from that a QueueSession. Thus, 30 QueueConnections are needed and only one QueueSession per connection. The connection pool maximum would need to be 30, but the session pool maximum would only need to be one.

These are two extreme scenarios, and the values in the examples do not take into account any other part of the application.

15.6 Security

When security is enabled on WebSphere Application Server V6, you must take certain steps for JMS applications that use the service integration bus to authenticate themselves to the bus to allow them to continue to use the messaging resources that they access.

- All JMS connection factory connections must be authenticated. You can do this in two ways:
  - The connection factory can have a valid authentication alias defined on it.
  - The JMS application can pass a valid user name and password on the call to ConnectionFactory.createConnection(). An ID passed in this way will override any ID specified in an authentication alias on the connection factory.
- All activation specifications must have a valid authentication alias defined on them.

Note: If a connection factory is looked up in the server JNDI from the client container, any authentication alias that is defined on the connection factory is unavailable. This prevents unauthorized use of an authenticated connection factory.

JMS Clients outside of the server can provide a user name and password on the call to create connection, or if the client is a J2SS client application that is running in the WebSphere application client environment, it is possible to define an authenticated connection factory resource in the EAR file.

Any user that authenticates as a valid user to WebSphere will, by default, have full access to the bus and all the destinations on it. It is possible to configure destination and even topic specific authentication requirement if you wish to do so. Refer to *WebSphere Application Server V6 - Security Handbook*, SG24-6316 for details.

Every bus has an optional inter-engine authentication alias which can be specified. If this property is left unset, then it will be ignored. However, if an alias is specified and security is enabled, then the ID is checked when each messaging engine starts communications with other messaging engines in the bus. This authentication provides additional security to prevent hackers who are pretending to be another messaging engine in the bus.

16

Planning for Web services

This chapter provides guidelines to help administrators understand the advantages of implementing Web services in their architecture. Web services are how service-oriented architecture (SOA) has been implemented in J2EE. This chapter describes Web services and what considerations you should take when planning for their usage on a WebSphere platform. It also introduces the changes that have been made as a result of J2EE 1.4 and how those changes affect the use of Web services in WebSphere. This chapter addresses the responsibilities of an administrator to deploy a Web service, and the considerations when deploying them to an intranet or Internet audience. It contains the following sections:

- Overview of Web services
- Extensible Markup Language
- SOAP
- Web Services Description Language
- UDDI and ebXML
- J2EE 1.4
- Web services in Java: JAX-RPC
- Enterprise Web services: JSR-101
- UDDI V3
- Java API for XML registries (JAX-R)
- WebSphere integration with Web services
- Benefits to implementing Web services

16.1 Overview of Web services

SOA is an architecture concept, an approach to building enterprise applications that focused on services (or loosely-coupled components) that can be composed dynamically. Alternatively, Web services, is one approach to building an SOA. Web services provide a standard implementation for SOA and that is the support that WebSphere Application Server provides.

Why are Web services so important? SOA is an important trend in the IT community. With the SOA approach to application architecture, existing applications can be converted to services which can be consumed by existing applications or new ones. As the architecture grows and more applications are added to this portfolio, they can be orchestrated together using what WebSphere calls *Process Choreography*. With Process Choreography, businesses are able to better react to changes in the business environment, such as the introduction of a new partner or supplier, shifts in the business model, or the streamlining of several application services into one.

Web services are Web-based enterprise applications that use XML-based standards to exchange data with calling clients. J2EE 1.4 contains the APIs for developing and deploying Web services in a managed environment.

16.2 Extensible Markup Language

Extensible Markup Language (XML) is a cross-platform, extensible mark-up language that is used to represent data. When XML data is exchanged between two parties, the parties can create their own tags to describe the data in their system, define schemas for mapping the data, and use stylesheets to manage the display and handling of the data. XML is the foundation of Web services. Specifications, such as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL), are derived from XML.

XML foundations

The following is an example of an XML document:

Example 16-1 Anatomy of an XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseOrder>
<shippingAddress>
<name>Doe</name>
<street>123 Newberry</street>
<city>Cary</city>
<state>NC</state>
<zip>27511</zip>
</shippingAddress>
<item id="123" description="Bunny, large, dark chocolate" quantity="5"/>
</purchaseOrder>
```

All XML files start with a prolog. It is essentially the same as an HTML header, except that it is delimited by <? and ?>. Standard information in the prolog includes the version, encoding, and whether the XML document is stand-alone or whether it references an external file. In Example 16-1 on page 461, the root level element identifies the beginning of the structure of the document, purchaseOrder. This element is used to describe all the other elements in the document. If applications in an enterprise want to use the same data, it is necessary to come to an agreement on the tags that are used to describe this data.

As the parser reads an XML file, each element in the file has its own job. Not above, that the content of the shipping address tag is contained entirely within the shipping address start tag and end tag. It is this ability for one tag to contain others that lets XML represent hierarchical data structures.

XML uses infosets to define the format of an XML document. A common reference is that all XML documents must be well formatted and valid. Infosets provide a way of describing how a document can be well formed. the infoset is a recommendation offered by the W3C to provide a consistent set of definitions for use in the other specifications that need to refer to the information in a well formed document. According to this recommendation, an XML document has an info set if it is well formed and satisfies the name space constraints.

XML namespaces

When defining an XML document, name conflicts can often occur. What happens when two XML elements share the same name but are not meant to be processed the same way? It is necessary to have a way to define the same name that is used in the different contexts. W3C defines a simple method for qualifying element and attribute names in an XML document. It associates them with namespaces that are identified by Uniform Resource Indicator (URI) references. For example, a namespace definition might look like that shown in Example 16-2.

Example 16-2 Example of defining an XML namespace.

```
...
<msg:message xmlns:msg= “http://messages.com/msg”>
...
```

The use of the namespace is declared using a family of reserved attributes. The attribute defining the namespace must be either `xmlns` or have an `xmlns` prefix. By providing a qualified name in an XML namespace, it is now possible to reduce the risk of running into name conflicts. With this concept, it is now easier to more fully qualify the names used in the XML document against a URI.

Using a qualified name can sometimes feel as though each element is being over qualified. As namespaces begin to be used, is can seem as though qualifying each element can be overwhelming. By using a default name space it is possible to avoid having to provide a qualified name for each element. You can set a default context that can be used.

XML schemas

As mentioned earlier, XML documents are to be well-formed and valid. InfoSets, as discussed in a previous section, are a way to ensure that documents are well formed. By defining an infoset, a description of the document structure in used to ensure the structure is adhered to by those wanting to use the same data.

The difference between infosets and XML type definitions is that infosets check for well-formed XML documents while type definitions verify the validity of the document. There are two ways to implement validation for XML documents: document type definitions (DTDs) and XML schemas. DTDs are the older mechanism for providing validation. The main limitation of DTDs is that it is only possible to tell that the element has data, not what kind of data to be used. With XML schemas, the concepts that are provided with DTD are used and then enhanced with additional data typing features. This data typing features is now described to the parser using XML.

An XML schema is a document that consists of a set of tags that declare the structure of other XML documents. Example 16-3 shows an example of a schema.

Example 16-3 Example of an XSD file.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xsd:element name="note">
    <xsd:complexType>
      <xsd:sequence>
   <xsd:element name="to" type="xsd:string"/>
   <xsd:element name="from" type="xsd:string"/>
   <xsd:element name="heading" type="xsd:string"/>
   <xsd:element name="body" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

At the top of the example, you see the standard prolog for the XML document as well as the declared namespace. The <xsd:schema> element indicates that the elements and data types that this schema uses come from this schema definition. It also identifies the prefix to be used by the qualifying elements in this schema.

Schemas can help to define standard data types, either simple or complex. However you can also use schemas to define custom data types and define facets or restrictions as to how the data is to be defined. Simple types map to what most developers know as simple of primitive data types with known precision. Complex types tend to combine multiple simple types. For example, a string and an int are simple types, but a “customer” might contain multiple strings and ints and other data types. The XSD or XML schema definition allows the definition and description of the data to be used.

XML parsers and tools

There are two APIs that provide standard ways of accessing XML files: document object model (DOM) and simple API for XML (SAX). The DOM parsing dynamic allows the application to manage the parser. The parser takes in the XML and creates a DOM tree. It then accepts change requests from the application and generates a new document as output. The application loads the parser, reads in the XML based on the schema, and the API then allows the application to read, write, or update the DOM tree.

SAX on the other hand provides read-only access. SAX is similar to DOM, but SAX does not allow the writing or changing of the XML document to be parsed.

For interoperability with Microsoft and other platforms, DOM must be used, and this is the standard in WebSphere.

There are some specific tools that are provided through the WebSphere platform to help in the building and maintaining of XML. IBM Rational Application Developer contains perspectives for dealing with this type of document type. The XML perspective contains wizards and tools for building XML, XSD, and DTDs as well as other tools to help with Web service development and deployment.

16.3 SOAP

This section describes SOAP as the main messaging format that is used for Web services. When considering Web services, SOAP is a major part of that implementation. This messaging format helps to provide a consistent means of getting data between providers and requesters of Internet services.

SOAP handles the transmission of data between clients and the Web services. This data is transmitted over HTTP to enable an interoperable exchange between clients and servers regardless of their platform or location in the network. The common term used to represent this protocol is SOAP over HTTP or SOAP/HTTP.

SOAP is a light-weight XML standard for distributed messaging. SOAP is considered light-weight because it does not include many of the features of existing distribution technologies like CORBA or RMI. The SOAP specification defines a structured message format and processing model that separates the application and infrastructure related concerns. It also provides an error handling mechanism through SOAP fault messages.

Message exchange patterns

The discussion of message exchange patterns (MEPs) comes down to how the information is organized in the body of the SOAP message. remote procedure call (RPC) exchanges carry explicit information identifying the operation being requested. The document-style carries an XML document that represents a business noun with the expectation that the receiving service is able to do something with the message.

The interpretation of the XML is based on an agreement between the communicating parties. The SOAP specification does not address the nature of that agreement, WSDL represents that agreement. The binding of a SOAP service often indicates the physical request-response characteristics of the service. For example, a SOAP/HTTP binding is inherently request-response due to the requirement that the HTTP request must have a response. The concept being discussed here is more abstract in that it addresses the logical nature of the communication between participating SOAP nodes.

A common planning question is which is better to use, RPC or document style? Document oriented services are noun focused rather than verb focused and tend to minimize chattiness. The operation being invoked is typically inferred from the document being sent in the context of the receiving service. For example a purchasing service might infer that a purchase is being made when it receives a PurchaseOrder and it can infer that an update to a customer's credit information is appropriate when it receives a CustomerCreditInfo message. Contrast this with an Accounting service that might infer the need to update accounts payable when receiving a PurchaseOrder.

Such services deal with business concepts like bank accounts, purchase orders, and so forth rather than granular items like bank account balance. Document-oriented services tend to be decoupled from any particular workflow but are able to participate in many. Document-oriented services are driven by business workflow and not by existing implementation artifacts. Document-oriented services often have a single entry point rather than a set of operations more common to an RPC-style service.

RPC-style services are typically the opposite. They are procedural, and the operation being invoked is explicit in the message content, for example a purchase (item). RPC-style services tend to be chattier and tightly coupled to existing implementation artifacts requiring the caller to become coupled to the interface of that implementation.

The SOAP processing model

At the application level, SOAP message exchange occurs between the initial sender of a message and the ultimate receiver. However, the SOAP processing model allows for the traversal of intermediaries — SOAP nodes that handle Quality of Service (QoS), infrastructure, and other messaging related operations that are not application specific. Common intermediation operations include: message logging, message routing, message authentication and authorization, and so forth.

Intermediaries can be deployed at the requestor/provider locations or on SOAP engines somewhere in between. They are often deployed for service management and QoS reasons. In general, intermediaries should not alter the meaning of the message body or influence the business semantics of the message.

Figure 16-1 illustrates a SOAP processing model.

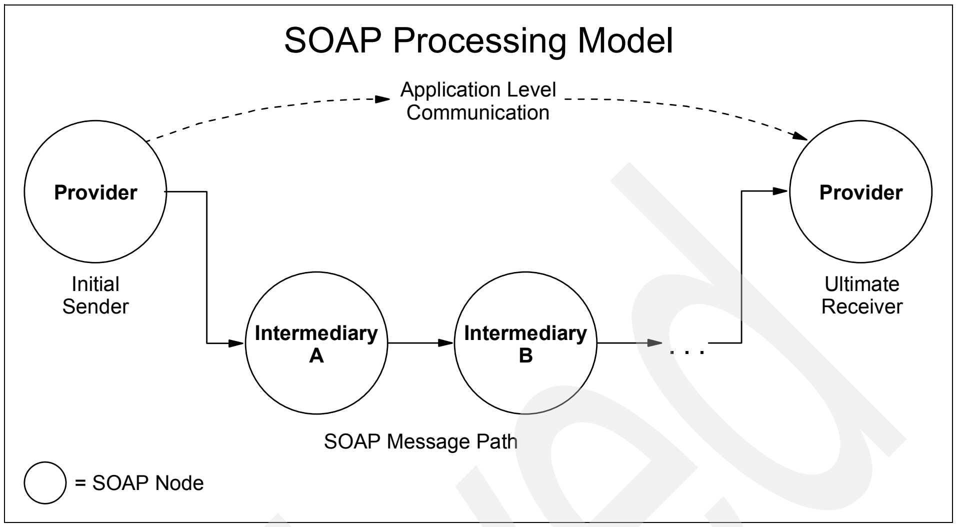


Figure 16-1 SOAP processing model

Components of a SOAP message

A SOAP message can contain an envelope, a body, and a header. The envelope is the top-level element in a SOAP message. The envelope namespace identifies the SOAP version for the message. The body is a required part of the soap envelop. It carries the payload of the SOAP message. The header element is optional. Each of the child elements is a header entry and is generally intended for a specific SOAP intermediary. SOAP faults can also be defined to represent failures that might occur in the transport or reading of the message body.

SOAP encoding

SOAP encoding provides a means of encoding instances of data that conform to the defined data model. SOAP has an optional encodingStyle attribute that can be used to describe the encoding mechanism for the message. The Web Services Interoperability (WS-I) discourages the use of the encoding style due to ambiguities in the specification that leads to interoperability problems. If this attribute is not specified, the style is literal.

When it becomes necessary to take a SOAP data model and write it out as XML, that is when SOAP encoding is used. In SOAP 1.1 the encodingStyle attribute is used to describe what to do when writing out a data model. Each outgoing edge becomes an XML element, which contains either a text value or additional sub-elements. Example 16-4 shows an example of an XML encoding.

Example 16-4 SOAP encoding example.

```
<product soapenv:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
   <sku>12345</sku>
   <name>Business Widget</name>
   <type>finance</type>
   <desc>Financial Application Widget</desc>
   <price>200.00</price>
   <inStock>245</inStock>
</product>
```

SOAP engine

A SOAP engine is bundled with WebSphere Application Server. This only supports the IBM branded engines of which there are two:

- An IBM version of Apache SOAP that is based on Apache SOAP Version 2.3
- The IBM SOAP engine

The IBM SOAP engine is similar architecturally to the Apache AXIS SOAP engine but uses parsing to improve performance. The IBM engine is the preferred engine for new deployments.

The job of the SOAP engine is to receive a message from a transport, check the SOAP semantics, process the SOAP headers, deserialize the message, and then route the message to the appropriate service implementation. After the service has be received and handled, the engine will then serialize the response, process the response headers, and then return the response over the transport.

16.4 Web Services Description Language

Web Services Description Language (WSDL) is a standards-based XML format that is used for describing network services. This description commonly includes the name of the service, the location of the service, and different ways to communicate with that service.

There is some debate over how much WSDL architects and developers need to understand because it is generated and consumed by tools. However, the reality is there can be a need to edit what is generated to achieve interoperability with other vendors' SOAP engines. Also, WSDL is the service definition that you are publishing. So, if you provide the interfaces to other applications and partners, you want to design the interface independently of your implementation.

WSDL provides a standardized way for service providers to describe to others the basic format of requests to their systems. It is the Web service contract between the client and the server.

16.5 UDDI and ebXML

Additional standards are available to help businesses publish their products and services over the Internet. Standards such as Universal Description, Discovery, and Integration (UDDI) and electronic business using Extensible Markup Language (ebXML) make it possible for businesses to make information readily accessible to clients around the globe. The flexibility of having services listed in either registry type encourages easy access to any service a business wants to expose.

UDDI

UDDI job is to solve the problem of publishing and locating services. The UDDI specification defines the way that services will be registered and located in an enterprise. UDDI itself is a Web service that allows you to access the registry services through SOAP. UDDI defines an information model that can be used by businesses to register the services they want to expose as well as serve as an inquiry API for searching the services that are available in the registry. UDDI also provides a mechanism for updating the registry and serve as a replication directory service for cooperating servers.

ebXML

ebXML is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes.

ebXML provides an alternative to UDDI that gives access to a different level of services. For further details about ebXML, see:

`http://www.ebxml.org`

16.6 J2EE 1.4

J2EE 1.4 had provided some new APIs to allow enterprise level support for Web services. These APIs that are specific to Web services include:

- Web services 1.1

  J2EE provides full support for the basic profile. The basic profile standardizes Web services implementation across vendors implementing J2EE. The difficulty with previous versions of Web services was that when it came time to implement them, there was a tight coupling between the client and the server. The basic profile attempts to dispel this pattern, and provide another layer for abstraction, for example Web Services Invocation Framework (WSIF).

  The WS-I organization is an industry consortium whose primary goal is to promote the adoption of Web services technologies. The WS-I created a definition that defined several conformance target to help support interoperability when using Web services. This definition is known as the WS-I Basic Profile 1.0. This profile defines three conformance targets. One is the use of SOAP messages, WSDL documents, and UDDI entries as a standard. The second is the definition of roles to participate in a Web service framework, a instance (producer), a consumer, and a registry. The third defined the need for a message sender and a message receiver. When all vendors conform to this profile, interoperability becomes faster and easier.

- JAX-RPC 1.1

  Defines the specification for accessing Web services through clients as well as defining Web services end points. JAX-RPC also describes how a Web service can implement message handlers and control message requests and responses.

- SAAJ 1.2

  SAAJ 1.2 is the API used to manipulate SOAP messages. This specification is used by JAX-RPC to provide access to XML fragments and the soap handler in the JAX-RPC message handler.

- JAXR 1.0

  The specification that supports client access to XML-based registries, such as ebXML or UDDI registries. JAXR has become the standard that is now required by Web service providers in WebSphere Application Server.

16.7 Web services in Java: JAX-RPC

The fundamental purpose of JAX-RPC is to make communication between Java and non-Java platforms easier, first by using basic profile conforming technologies (such SOAP, WSDL, and UDDI) and then by using a simple Java API that can be used to communicate to these technologies. The following sections describe the details of the specification and the goals behind the technology.

16.7.1 What is JAX-RPC

JAX-RPC is a specification that defines both the server-side and client-side programming models that are needed to provide the implementation for how clients are able to interact with Web services. The server-side programming models provide the APIs for creating EJBs to act as end points for communicating with services. The client side programming model provides Java applications a way to exchange SOAP messages with remote Web service end-points on any platform.

The latest version of this specification added support for WS-I basic profile at Version 1.1. This basic profile is used to guarantee service support across platforms that are compliant to this definition. The purpose of JAX-RPC is to provide support for multiple protocols bindings that are based on the XML infoset, and SOAP 1.2 messages are formed through XML infosets.

JAX-RPC is meant to be a protocol neutral specification, but it does require SOAP/HTTP. It also defines the mapping of WSDL and XML to Java, and vice versa, to help clarify the data mapping between service requesters and service providers. JAX-RPC specifies client APIs to invoke a remote Web service and a runtime environment for Java Bean as an implementation of a Web service. The specification defines a Handler class to act as a filter for executing certain tasks just prior or just after the service invocation.

JAX-RPC is designed for portability of code across multiple vendors. Any client or service implementation should be portable across any vendor that supports JAX-RPC.

16.7.2 JAX-RPC goals

One of the main goals of JAX-RPC is to formalize the procedure for making calls to Web services in an RPC-like manner. In order to facilitate this, another goal of the specification is to provide APIs for the client to use to invoke Web services. There is a need for the client to be able to map its data requests to the service being provided and JAX-RPC provides the mechanism for doing the mapping between Java and XML types. The final goal of this specification is to give you an interception point in front of the Web services, analogous to a servlet filter that is available in J2EE. It provides a way to intercept the request and response before and after the service has been invoked.

The type mapping framework includes the services that aids in the mapping that needs to be done between the client and the service. These services include a type mapping registry, a type mapping service, a serializer, and a deserializer as shown in Figure 16-2.

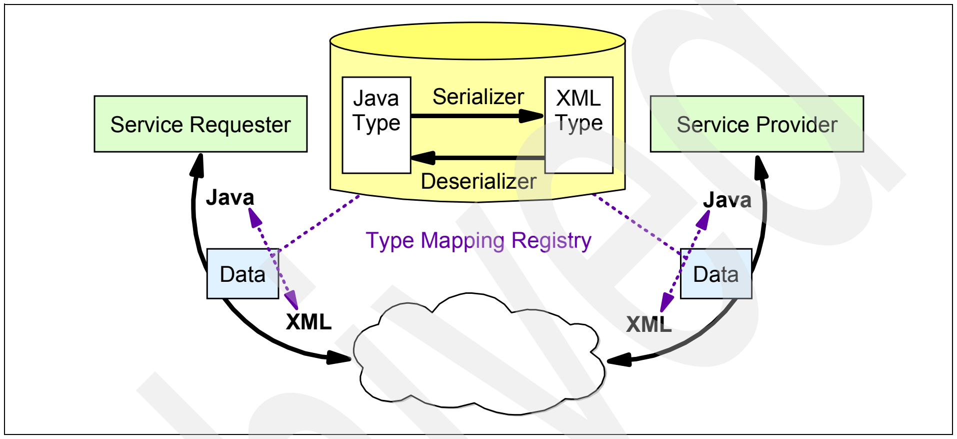


Figure 16-2 Type mapping framework

16.7.3 Handlers

Handlers provide a mechanism for intercepting the SOAP message and operating on header information. They can examine and potentially modify a request before it is processed by a Web Service component. The class can examine and potentially modify the response after the component has processed the request.

Handlers are conceptually similar to Servlet 2.3 Filter and can be provided for both the client and server. A handler must not change the SOAP message, operation name, number of parts in the message or types of the message parts.

Handlers allow the interception of the SOAP message before it reaches the Web service, as well as a handle to the SOAP message on the way back to the client.

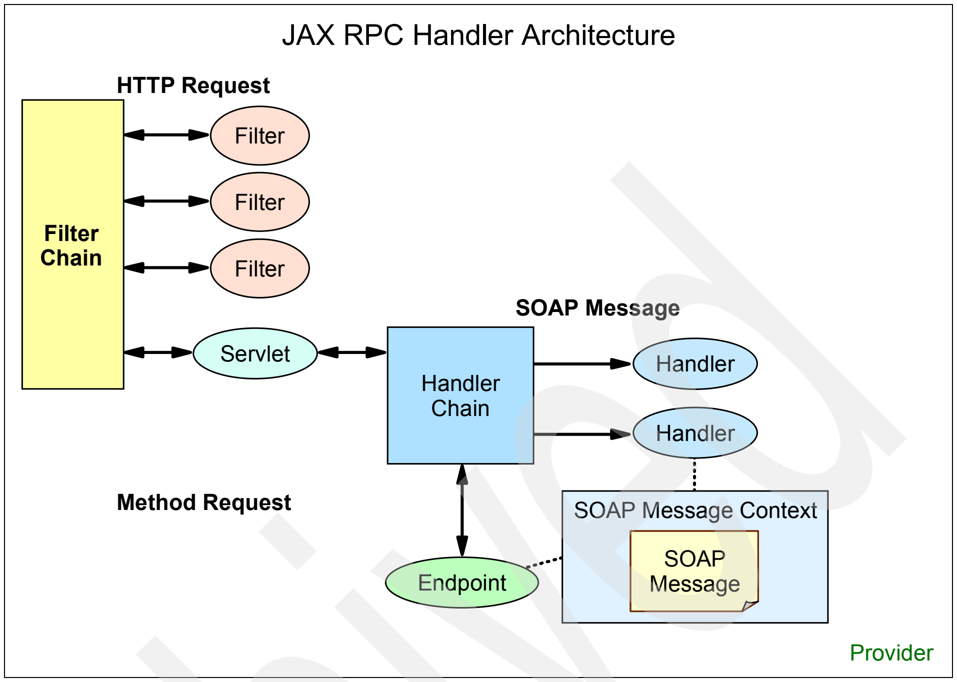


Figure 16-3 JAX-RPC handler architecture

16.8 Enterprise Web services: JSR-101

The JSR-101 specification has also been named Web services for J2EE or Enterprise Web services. The purpose of the specification basically fills the gaps that the JAX-RPC specification left open for use of Web services in a J2EE Application Server. The following section will describe the features, invocation styles and deployment packaging defined by this specification.

16.8.1 Features

The specification defines the deployment requirements for the application through the use of standard Web services deployment descriptors. It also defines a J2EE compliant deployment and packaging model for Web services on the server side and for the client.

The new deployment descriptors for Web services allow for declarative definition of the use of Web services in an architecture. There are also server-side programming model extensions that can help developers build and use the application server to deploy services of this type. The specification also suggests the use of a Stateless Session EJB as implementation of a Web service, known as a service endpoint interface (SEI).The SEI acts as the EJB's remote interface (or it must contain a subset of the remote interface's methods).

It does recognize the potential performance problems with using entity and stateful enterprise JavaBeans and, therefore, specifies that they cannot be exposed as a Web service. A client-side programming model is included to provide standardization between the service providers and the service requesters, and with this model, any J2EE container can act as a client to a Web service, such as EJB, servlet and JSPs, and application clients.

16.8.2 Client invocation styles

There are three invocation styles as defined by the client programming interfaces in the specification. The three invocation styles are static (generated) stub invocation, which is a java class that is statically bound to an SEI, WSDL port, and port component. This type of style is Web services work today. The second way to call a Web service is through a dynamic proxy invocation. This is a Java class that is not statically bound to the Web services and the client can dynamically obtain a dynamic proxy at runtime by providing the SEI. This services uses the JDK 1.3 dynamic proxy feature. The third option is the use of a dynamic call DII invocation. This uses an object provided by the RPC APIs. The javax.xml.rpc.Call object is instantiated and configured to invoke the Web service. This option is useful when a client needs dynamic, non-stub based communication with the Web service.

16.8.3 Deployment packaging

The deployment packaging for installing a Web service on a J2EE application server is defined through several components. Included in the packaging are port components (SEI and implementation) which are the interfaces mentioned earlier that serve as the communication entry point for talking to a Web service. Also, there is a specific Web services deployment descriptor in each respective archive (Web or EJB). For the EJB, the file is located in the EJB Module META-INF directory. The Web services deployment descriptor is in the Web Module WEB-INF directory. The final part of the packaging includes the information that the server needs to determine how to map the data types coming from the client in Java to the server interface defined in WSDL. This file is co-located with the module descriptor and is referenced by the Web service deployment descriptor.

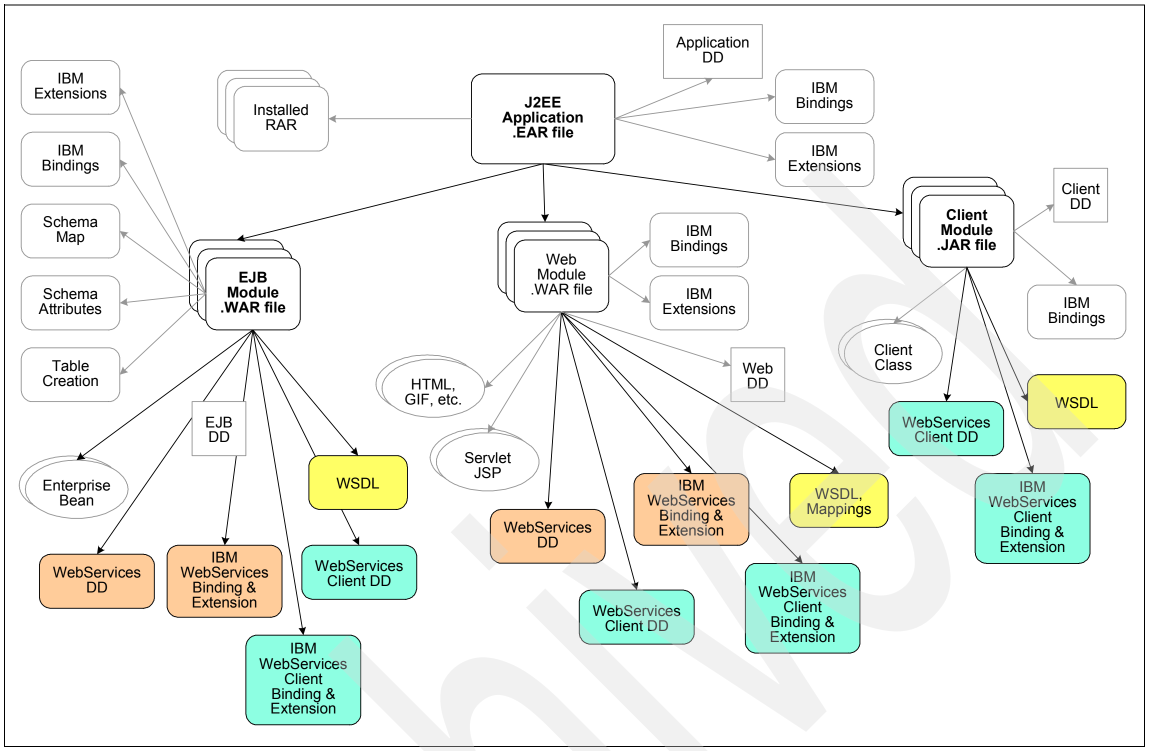


Figure 16-4 Example of a Web service enabled EAR file

16.8.4 SAAJ

SAAJ is mainly an API that is used by other APIs to accomplish rendering of literal XML snippits (SOAPMessage). It manipulates XML-based SOAP messages. The API can be used in a non-managed or stand-alone environment to create SOAP-based messages. It is also important to note that the SAAJ APIs are parents to those that are used in the messaging API that is provided by Apache's AXIS.

16.8.5 JAX-R

JAX-RPC and JSR-109 do not address how to remotely discover a Web service. There are many different ways that a developer can locate a WSDL file. For example, the developer could use a URL, UDDI, UDDI4J, or JAX-R. This API has been standardized as part of J2EE 1.4 and it provides a basic, standardized API for accessing XML registries (including UDDI).

16.8.6 Security

When planning for Web services, it often becomes important to think about the benefits and consequences of implementing security. The J2EE specification provides some new APIs to help in this implementation.

Java Authorization Contract for Containers (JACC) defines a contract between an application server container and a authorization policy provider. All J2EE containers are required to support this contract.

WS-Security is a specification that provides support for any possible security combination. However, at this time, there is no API exposed. The specification provides a way to add to the specifications through WS-Trust and WS-Secure Conversation. The main difference is in how the security service is leveraged in WebSphere. There is a caveat to this implementation that is important to consider for planning. If you have already deployed an application in WebSphere Application Server V5 and you want to migrate the application to WebSphere Application Server V6, you will have to make some changes to the deployment descriptors to take advantage of the WS-Security that was not available in previous versions. WS-Security is beneficial to the security architecture because it provides the ability to use the JCE framework, allows the encryption of any part of the SOAP message, and supports the use of pluggable tokens to enforce signatures and encryption.

16.8.7 Web services and performance

A common planning consideration is performance. Often the business processing takes much more time than the Web service processing. Thus, it is important to design the applications with this in mind. Some considerations when thinking about these situations are:

- Do not make unnecessary calls.
- Minimize the number of requests by caching frequent used data on requestor.
- Minimize information being exchanged.
- Send a key instead of an entire record.
- Send only changed data.
- Send most common data.
- Limit the number of network calls.
- Balance® the number of network calls versus sending larger amounts of data.
- Let business needs drive your design.

An architect should limit the number of calls that are made to avoid the network congestions often accompanied by these calls. An architect should also consider whether it is more efficient to get the data in a granular fashion or retrieve a large amount of data and parse through it at a later time. The trade-off is accuracy versus performance. If the data is retrieved in a large amount and parsed later, the data can change while in use. If the access is more granular, then the performance is degraded, based on the additional calls that are needed to get the information.

Another common performance concern is how much data is transferred. It is a better practice to send only the data needed, or only the data that has changed, in a request or response. This practice avoids sending data across the network that is not needed by the transaction.

As a design practice it is recommended that you minimize the number of network calls that are made. This helps in performance but limits the scalability of the application. As with most design practices, when considering performance, it is important to consider trade offs to get that better performance.

16.9 UDDI V3

UDDI is a registry that provides organizations with the ability to publish their services and have them searched by Web service consumers. These registries provide the details about business service types and allows Web service consumers to use that data to facilitate application integration. The type of data commonly held in a registry includes a listing of businesses or organizational units, the business services they provide, and where to locate that service.

UDDI is needed to provide access to Web services through the network. It is not required for building and using Web services. UDDI helps in the distribution of the required files for accessing a Web service, however it is possible to deploy and consume a Web service without the lookup through UDDI. This can be accomplished through direct communication with the Web service provider.

If a UDDI registry is to be used to deploy Web services, it is important to understand the impact of this choice. The following sections will cover the different considerations and aspects of the UDDI v3 implementation.

16.9.1 Issues UDDI solves

UDDI provides a broader business-to-business environment. It allows for a service provider to publish services to many different types of customers, with varying standards and protocols. It also provides a more valuable search service that connects businesses in a broader way. UDDI also provides a solution for business looking for an easier way to aggregate services, such as locating specific types of data, connecting with suppliers and business partners. One of the main benefits is that UDDI provides a standards based profile for all electronic services to include e-commerce, portal, and traditional Web sites.

16.9.2 Multi-registry environment

There are two basic registry topologies: a single-node topology and a multi-node topology. A single-node registry provides a single location for all registered services. A single-node registry differs from a multi-node registry in that a multi-node registry provides a mechanism for replication. If a business requires the ability to have multiple registries replicating against a main registry, the multi-node topology can be used. Each node, in either topology, is organized in a hierarchy using a nested model including business information, service information, and binding information. It is also categorized using white pages, green pages and yellow pages. White pages provide information about the business entries, contact information and business identification information. Green pages provide the technical information about how to bind to a specific service. Yellow pages provide the service of categorization, similar to the concept of yellow pages in a phone directory.

UDDI 3 provides the ability to publish services and data across registries. There is a root registry that keeps track of the main list of services through key values. The root registry then contains the master registry and is responsible for replicating the registry data to the other machines. This multi-registry support can also be used to support inter-registry publication and the use of registries across nodes.

UDDI 3 uses a multi-registry environment that supports the use of root and affiliate registries. This support provides some guidance on how to manage inter-registry associations. The root registries act as the authority for key spaces, and are used to delegate key partitions such that other registries can rely upon the root registry to verify and maintain the uniqueness of such key partitions. Registries that interact with the root registry are called affiliate registries. This is not the same as replication that was mention earlier, though replication is still supported.

16.9.3 UDDI keys

Previous versions of UDDI required that when keys were generated, they needed to conform to the UUID definition and needed to be assigned per node. Now with UDDI 3, a recommendation is that the use of domain name system (DNS) names be used. This naming convention would include the domain and key-specific string for identifying the registry. The benefit of this change is that the keys are easier to administer and understand.

Affiliate registries rely upon the root registry for delegation of unique key partitions. Data can then be shared between and among the root and affiliate registries with the assurance that each given key partition is unique. This version allows publishers to take advantage of a federated environment, which is beneficial in the situation where a business want to publish a service more than once to encourage broader exposure and use. So, publishers can provide their own keys, as opposed to the previous version where keys were generated by the system.

Keys are now implemented as URIs. This change allows publishers to use the concept of vanity keys which supports a user-defined name instead of a UUID that would be generated for pure uniqueness. There are 3 formats that can be used for generating this key. One is the UDDIKey, which is the key format that is compatible with the UUID format of previous versions of UDDI. The second is a DomainKey, which would represent an established domain name. Additonaly the third is a DerivedKey which is derived from another key. These new options allow publishers to generate keys that are now easier to manage, remember, and use.

16.9.4 XML digital signatures

The security model in UDDI 3 defines a set of node and registry policies and respective implementations for those policies. Now that UDDI 3 can use multiple registries, is it more important than ever to be able to determine and control the reliability of the data. UDDI 3 adds the support for digital signing of all core data types, through XML digital signatures. By digitally signing data in the registry, publishers can now get around any spoofing of their data that is being requested by inquirers. If the data is changed in any way, it breaks the signature, which allows both the publisher and inquirer to feel confident in the data's integrity.

16.9.5 UDDI policies

In earlier versions, the UDDI specification mixed policy issues and UDDI Business Registry (UBR) implementation concerns. UDDI V3 separates these and expands the role of policy by prescribing a set of policies registries must address. It establishes a set of policy abstractions for which corresponding policy rules must be defined by the registry.

The registry implements policy decision points to refer to the policy rules and policy abstractions that have been specified. Each node then uses a policy enforcement point to implement the policy settings defined by the registry. There are many different policy categories that UDDI 3 supports, including policies such as data custody, replication, and delegation.

16.9.6 Discovery features

UDDI 3 also enhanced the mechanism to search for services in the registry. It provides more powerful and efficient queries through the nesting of sub-queries and the ability to use find_tModel within a single query. The tooling for providers has been enhanced to allow control over the number of round trips to the registry, as well as the ability to narrow in on the type of services being searched. New and extensible find qualifiers and sort orders have been added to enhance the search process. The support for these extensible qualifiers are provided through unicode collation algorithm (UCA) and sorting orders for specific languages or locales.

An additional enhancement was to extend wildcard support to use SQL-like approximate matching and allow the use of wild cards for searches that include things other than names. UDDI 3 also allows you to retrieve result sets in chunks, easing the management of large datasets.

16.9.7 Support

In WebSphere Application Server V6, all of the mandatory parts of the UDDI 3 specification are added as well as some of the optional services. WebSphere also enhanced UDDI 3 with some services that go beyond the UDDI 3 specification. The following is a list of those optional additions as well as any enhancements that are not part of the UDDI specification:

- V1 and V2 inquiry and publish APIs.
- Ownership transfer (intra-node custody transfer).
- Additional functionality.
- GUI for inquiry and publication.
- Administrative and management interface and GUI.
- UDDI v3 client for Java.
- V2 EJB interface, and UDDI4J, for backwards compatibility.

16.10 Java API for XML registries (JAX-R)

JAX-R is a client side API used for accessing different kinds of XML business registries. The main use is for accessing UDDI or ebXML registries. In some ways, this API is analogous to JDBC. JDBC provides an API for accessing relational data, where JAX-R provides a vendor neutral API for accessing any UDDI registry. The following section will describe the specification and its intended usage.

16.10.1 What is JAX-R

JAX-R provides multiple clients the ability to work with many different registry types. Because there are a few registry types accepted in the industry, an architect does not want to have to implement a solution for each registry needed in the enterprise. Instead the JAX-R registry APIs can be used to provide a single interface to all registries used in the system (such as UDDI or ebXML).

In JAX-R, capability profiles are used to categorize the functions available to an implementer. A JAX-R provider must select a capability profile to indicate the level of functionality, or methods, they are interested to use.

16.10.2 JAXR compared to UDDI Java APIs

JAX-R does not map directly to UDDI, so a more precise mapping might be desired. However, to get this level of mapping you will have to use an API that is not accepted as part of the J2EE 1.4 standard. There are a couple ways to implement this. One is UDDI4J which maps directly to UDDI, and provides a more complete solution for accessing the UDDI programmatically without the limitations of JAX-R.

Another option is the IBM Client for IBM Java Client for UDDI 3. Because UDDI4J can only interface with UDDI 2, there will be compatibility problems when an architecture takes advantage of WebSphere Application Server V6, which includes UDDI 3. Therefore, IBM has included a special API for accessing UDDI 3, and it is called the IBM Java Client for UDDI 3. Ultimately, JAX-R provides portability across all of these varieties and levels of support in the application server.

16.10.3 Using JAX-R API

The JAX-R API provides classes for accessing a XML-based registry. The process of communicating to a registry is similar to that is done in other access-type API's like JNDI or JDBC. To optimize the connection management, the API provides a ConnectionFactory. This class optimizes the performance of searching the registry though the use of a manager service that controls the connections that are available to the registry. Methods or functions that are available through this class include the ability to initialize the connection manager as well as a way to request a connection to the registry. Example 16-5 on page 481 shows an example that uses the JAX-R API.

Example 16-5 Using the JAX-R API.

```
// specify the ConnectionFactory implementation classname
System. setProperty(" javax. xml. registry. ConnectionFactoryClass", "com. ibm.
xml. registry. uddi. ConnectionFactoryImpl");
// instantiate a ConnectionFactory object
ConnectionFactory connectionFactory = ConnectionFactory. newInstance();
// instantiate a Properties object and add the Inquiry and Publish URLs
Properties props = new Properties();
props. setProperty(" javax. xml. registry. queryManagerURL", "http://
localhost: 9080/ uddisoap/ inquiryapi");
props. setProperty(" javax. xml. registry. lifeCycleManagerURL", "https://
localhost: 9080/ uddisoap/ publishapi");
// add the properties to the ConnectionFactory
factory. setProperties( props);
// create the Connection
Connection connection = factory. createConnection();
```

There are also API methods to create, update, and delete registry objects, when a connection has been established. You can have the objects return one registry item of a collection. This API is similar to the UDDI publisher's API. Another object that is instrumental in using the registry is the query manager. This is the object that controls the ability for requesters to look up resources in the registry.

With JAX-R, security can sometimes be something for which to plan. The APIs provide a mechanism for implementing security. This can be done by calling the setCredentials method. You can use this method to set the credentials for a user.

16.11 WebSphere integration with Web services

There are many products in the WebSphere platform that aid an architect in implementing Web services. Some of these products come with the packaging. Others need to be acquired separately. This section outlines some common products that are used when implementing Web services with WebSphere Application Server.

16.11.1 Web Services Gateway

The Web Services Gateway is a run-time component that provides configurable mapping based on WSDL documents. It maps any WSDL-defined service to another service on any available transport channel. It is usually deployed at the firewall and has access to internal services. The Web Services Gateway provides the following features:

- Service mapping

  The primary function of the Web Services Gateway is to map an existing WSDL-defined Web service to a new service that is offered by the gateway to others. The gateway thus acts as the proxy. External services are imported into the gateway and made available to the enterprise as proxy services internally. Likewise, internal services are imported into the gateway and made available as proxy services externally. These services are also published to the relevant UDDI directories where required.

- Export mapping

  An internal service can be exported for outside consumption. Given the WSDL file, the gateway generates a new WSDL file that can be shared to outside requestors. The requestors use the gateway as the service endpoint.

- Import Services

  Similarly, an external service can be imported and made available as an internal service. This helps the internal service requestors to invoke the service as though it is running on the gateway.

- Transformation

  A request for a service can originate on one protocol, but the service can be invoked in some other protocol by using the transformation function. An internal service that is available on SOAP over JMS could be invoked using SOAP over HTTP.

- UDDI publication and lookup

  The gateway facilitates working with UDDI registries. As you map a service for external consumption using the gateway, you can publish the exported WSDL in the UDDI registry. When the services in the gateway are modified, the UDDI registry is updated with the latest updates.

- Security and management

  Provides a single point of control, access, and validation of Web service requests.

16.11.2 Web services and WebSphere Business Integration

WebSphere Business Integration provides customers with a way to integrate their systems together seamless. WebSphere Business Integration is composed of five targets.

- The first is best of breed process. Companies can, through the use of WebSphere Business Integration accelerators, get processes that are provided by the best in the industry and integrate those processes with their systems.

- Second, WebSphere Business Integration provides process templates for increasing the availability of a company's existing business processes and turning existing assets into exposable services.
- Third, there are specific business adapters that are provided with packaged applications. These adapters also aid an organization in the integration of services across their enterprise.
- Fourth, custom adapters can be built using the products APIs to enable in house applications to participate in the enterprise.
- Finally there are user-based dash boards that allow a company to easily manage and monitor the effectiveness of their processes.

Figure 16-5 shows this relationship.

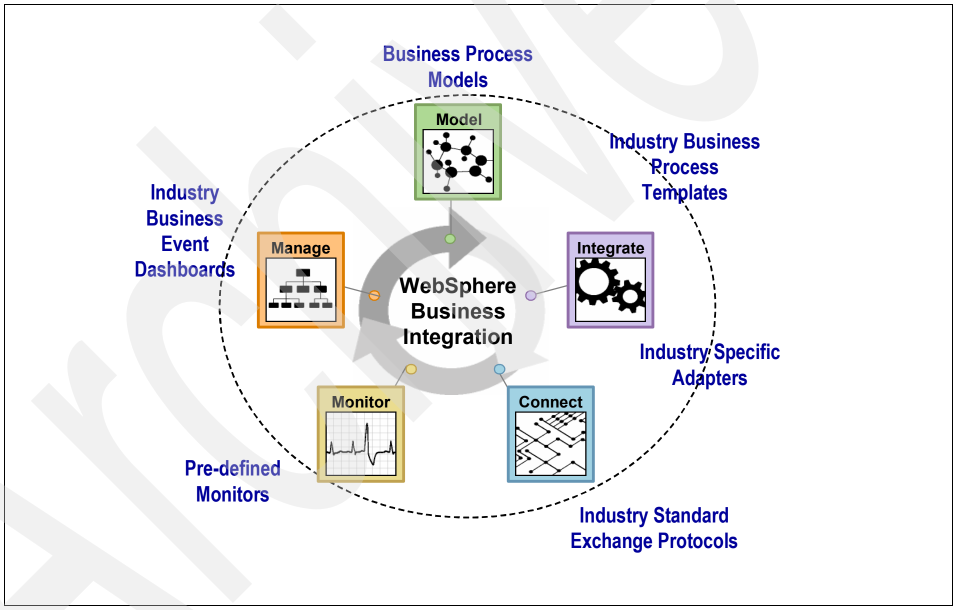

Business Process Models
Model
Industry Business Process Templates
Industry Business Event Dashboards
Manage
WebSphere Business Integration
Integrate
Industry Specific Adapters
Monitor
Connect
Pre-defined Monitors
Industry Standard Exchange Protocols

Figure 16-5 WebSphere Business Integration

16.12 Benefits to implementing Web services

Web services have become a standard way to access services across the network. This support has been standardized through the J2EE 1.4 specification and the WS-I organization. This section outlines why the industry has turned to Web services and the advantages that Web services have over other technologies.

16.12.1 Web Services as part of SOA

In current development efforts, application developers are focused on building functional units of code for one purpose only. One benefit of moving a development paradigm to Web services is that the development effort now is focused on building service based logic. This service can be exposed to many potential business contexts. Because SOA is focused on building applications with well-defined interfaces, applications can now be loosely-coupled and implemented under many different service requirements.

This approach eases the task of product integration and the integration with existing business systems and processes. Another benefit is that through the design of a SOA architecture, an organization needs to bring the IT and business strategies closer together. IT can think about the overall business strategy when deciding what services to expose and how. SOA encourages a closer modeling of the business processes so that isolation of the of different parts of the system can be accomplished, and therefore changes to this system can also be easily identified and implemented. The main benefit to SOA is the encouragement it provides for building the best choice of services.

Here are some questions to ask if you are considering implementing Web services as part of an SOA:

- Which business process will provide the biggest benefit if they are built are re-engineered into Web services?
- Which service interface design would yield the most to the organization?
- Which enterprise applications would be best wrapped as Web services and provide and easy entrance point for other applications to be refactored over?
- Which business process will provide the biggest benefit if they are built or refactored as services?
- What is most useful to important service requestors?
- Which existing IT investments can leverage this new service pattern?

The benefits to using Web services with J2EE and WebSphere include security, transaction control across distributed resources, and the ability to have the Web services leverage the container services available in J2EE.

16.12.2 Security

J2EE provides the ability for individual EJB methods to identify security constraints through method permissions. J2EE also provides end-to-end security, which means that no matter how the client is secured (via the Web container's HTTPS or WS-Security), fine-grained access control can still be used to access the EJBs that are interfacing with the deployed services. Because the security in J2EE is tightly integrated in to the runtime of the server, this security model also extends to cover Web services which are now a part of J2EE 1.4.

16.12.3 Transaction control

Another benefit is the transaction service provided by the container in WebSphere Application Server. The transaction service supports the ability for a transaction to be managed across multiple, distributed resources. Even if an architecture does not take advantage of the WS security specifications like WS-AtomicTransaction or WS-Coordination, it is still possible to use this service to synchronize the back end resources and maintain their transactional context.

16.12.4 Container services

One of the services that can offer some benefits is container managed persistence. As applications become larger and more complex, architects and planners are beginning to realize the benefit of being able to save time and effort on a model that is managed by the application server. Container managed persistence offers a framework that provides a programming model and tools to build a reliable infrastructure. These tools automate the process of writing or building the persistence logic into an application. This flexibility makes it quick and easy to build an application using Web services that have a database as it main functionality. This frees the architecture for depending on a single database vendor or type.

17

Planning for security

WebSphere Application Server provides security infrastructure and mechanisms to protect sensitive Java 2 Platform, Enterprise Edition (J2EE) resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability. This chapter covers the features that WebSphere Application Server provides and considerations that you need to take to plan for a secure environment. This chapter contains the following sections:

- Why do you need security
- Security fundamentals
- J2EE security
- Programmatic security

17.1 Why do you need security

The fundamental reason for having security is to keep intruders out and to prevent unauthorized users from gaining access to your systems. There are many reasons for intruders to gain access to your systems. They could be hackers, competitors, or even your own employees. Intruders might want to gain access to information they should not have or to alter the behavior of your systems.

Figure 17-1 illustrates the building blocks that comprise the operating environment for security within WebSphere Application Server.

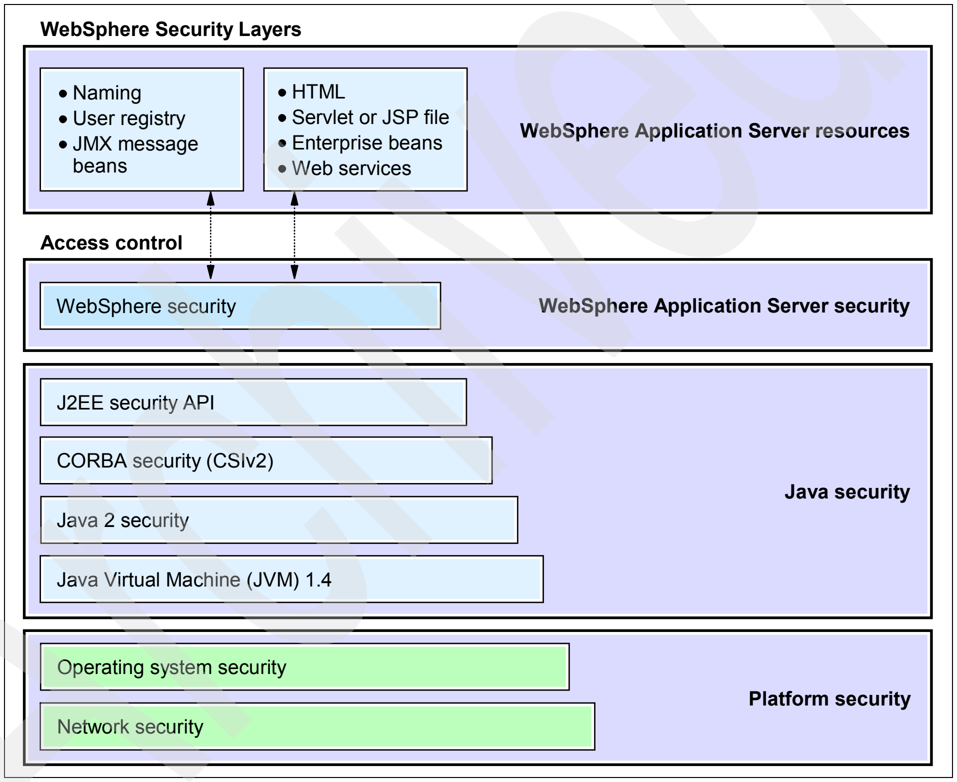


Figure 17-1 WebSphere Application Server security layers

Network security

The Network security layers provide transport level authentication and message integrity and encryption. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL) and HTTPS.

Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.

Operating system security

The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. These services include the file system security support that secure sensitive files in the product installation for WebSphere Application Server. The system administrator can configure the product to obtain authentication information directly from the operating system user registry.

JVM 1.4 security

The JVM security provides a layer of security above the operating system. Standard Java security is provided through the Java Virtual Machine (JVM) used by WebSphere Application Server and the Java security classes.

Java 2 security

The Java 2 security model offers fine grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource. WebSphere global security settings allow you to enable or disable Java 2 security and provide a default set of policies. Java 2 security can be activated or inactivated independently from WebSphere global security. The current principal of the thread of execution is not considered in the Java 2 security authorization. There are instances where it is useful for the authorization to be based on the principal, rather the code base and the signer.

CSIv2 security (CORBA)

Any calls made among secure Object Request Brokers (ORB) are invoked over the Common Security Interoperability Version 2 (CSIv2) security protocol that sets up the security context and the necessary quality of protection. After the session is established, the call is passed up to the enterprise bean layer. For backward compatibility, WebSphere Application Server supports the Secure Authentication Service (SAS) security protocol, which was used in prior releases of WebSphere Application Server and other IBM products.

J2EE security

The security collaborator enforces J2EE-based security policies and supports J2EE security APIs. APIs are accessed from WebSphere applications in order to access security mechanisms and implement security policies. J2EE security guards access to Web resources such as servlets and JavaServer Pages (JSPs) and Enterprise JavaBeans (EJBs) methods based on roles defined by the application developer. Users and groups are assigned to these roles during application deployment.

WebSphere security

WebSphere Application Server security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere Application Server security technologies and features to support the needs of a secure enterprise environment.

17.2 Security fundamentals

This section discusses two fundamental security concepts that WebSphere Application Server supports:

- Authentication
- Authorization

17.2.1 Authentication in WebSphere

The authentication mechanism in WebSphere Application Server typically collaborates closely with a user registry. When performing authentication, the user registry is consulted. A successful authentication results in the creation of a credential, which is the internal representation of a successfully authenticated client user. The abilities of the credential are determined by the configured authorization mechanism.

Although this product provides multiple authentication mechanisms, you can configure only a single active authentication mechanism at a time. The active authentication mechanism is selected when configuring WebSphere Application Server global security.

WebSphere Application Server supports the following authentication mechanisms:

- Simple WebSphere Authentication Mechanism (SWAM)
- Lightweight Third Party Authentication (LTPA)

Simple WebSphere Authentication Mechanism

The SWAM is intended for simple, non-distributed, single application server run-time environments. The single application server restriction is due to the fact that SWAM does not support forwardable credentials. If a servlet or enterprise bean in application server process 1, invokes a remote method on an enterprise bean living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, can cause authorization failures.

Because SWAM is intended for a single application server process, single signon (SSO) is not supported.

The SWAM authentication mechanism is suitable for single server environments, software development environments, or other environments that do not require a distributed security solution.

Because SWAM is only intended for single process settings, it is not applicable and supported in a topology involving WebSphere Application Server Network Deployment.

Lightweight Third Party Authentication

LTPA is intended for distributed, multiple application server and machine environments. It supports forwardable credentials and SSO. LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

The LTPA protocol enables WebSphere Application Server to provide security in a distributed environment using cryptography. Application servers distributed in multiple nodes and cells can securely communicate using this protocol. It also provides the SSO feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. The realm names on each system in the SSO domain are case sensitive and must match identically.

When using LTPA, a token is created with the user information and an expiration time and is signed by the keys. The LTPA token is time sensitive. All product servers participating in a protection domain must have their time, date, and time zone synchronized. If not, LTPA tokens appear prematurely expired and cause authentication or validation failures.

If the receiving servers share the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure it has not expired and the user information in the token is valid in its registry. On successful validation, the resources in the receiving servers are accessible after the authorization check.

All the WebSphere Application Server processes in a cell (cell, nodes, and application servers) share the same set of keys. If key sharing is required between different cells, export them from one cell and import them to the other. For security purposes, the exported keys are encrypted with a user-defined password. This same password is needed when importing the keys into another cell.

When security is enabled for the first time with LTPA, configuring LTPA is normally the initial step performed. LTPA requires that the configured user registry be a centrally shared repository such as LDAP or a Windows domain type registry so that users and groups are the same regardless of the machine.

In a WebSphere Application Server, Network Deployment environment, LTPA is the only option available.

17.2.2 Authentication process

Figure 17-2 shows the authentication process in WebSphere Application Server V6. As the figure shows, authentication is required for enterprise bean clients and Web clients when they access protected resources.

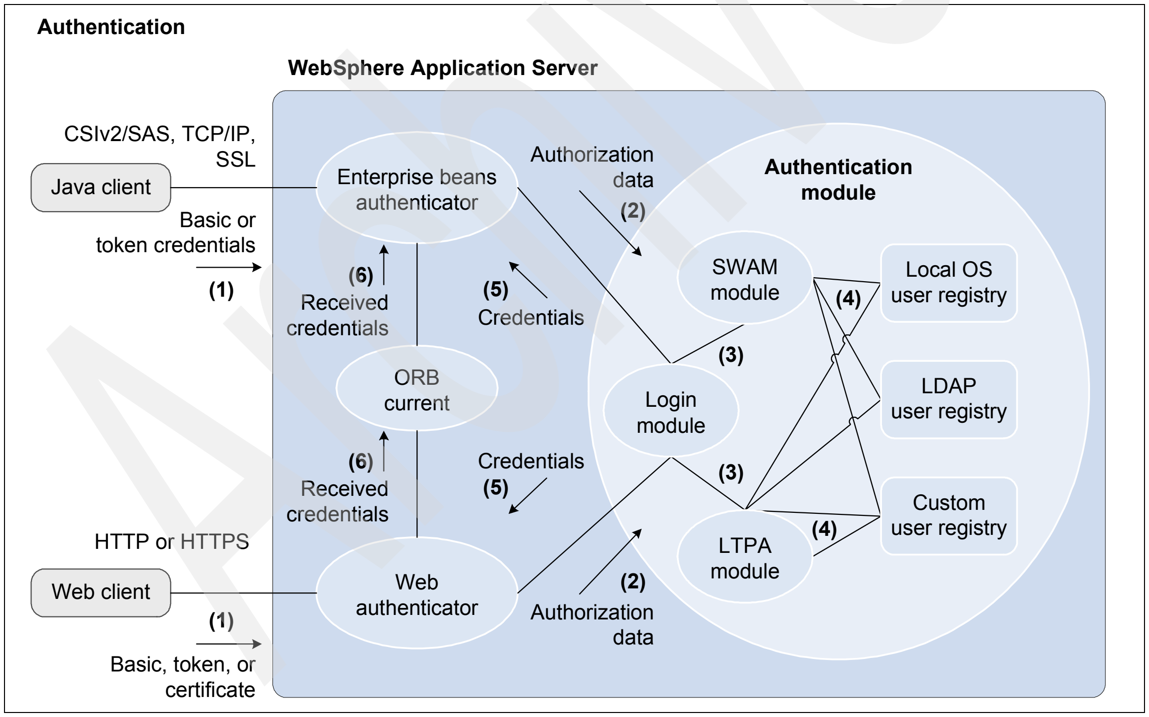


Figure 17-2 Authentication Process

As illustrated in the figure, authentication involves two primary cases.

- Web client authentication
- EJB client authentication

Web client authentication

When a security policy is specified for a Web resource and WebSphere Application Server security is enforced, the Web container performs access control when the resource is requested by a Web client. The Web container challenges the Web client for authentication data if none is present according to the specified authentication method, ensure the data constraints are met, and determine whether the authenticated user has the required security role.

WebSphere Application Server takes the authentication information, looks up the user's unique ID in the registry, and then verifies the password against the registry.

Securing Web components

Web components such as static HTML files, JSPs, and servlets can be secured using either the HTTP Server or WebSphere Application Server. When discussing securing Web components, there are two elements to consider:

- Static HTML components
- Servlets and JSPs

Static HTML components

Most Web servers can secure the files they serve. For example IBM HTTP Server can protect its resources using the following mechanism:

- HTTP basic authentication uses user identity in the network or the user ID and password the user submits. The authentication can also be made based on a combination of these elements.
- Digital certificate authentication.

WebSphere Application Server also has the ability to serve static content using the built in Web server. WebSphere cannot manage the static content stored in the Web server. It can only serve the content that is packaged as part of the Web module (WAR file). The Application Server Toolkit can be used to set up security constraints to protect static content for the Web application module.

Servlets and JSPs

WebSphere Application Server can secure dynamic resources such as servlets using role-based declarative security mechanisms. This means that the logical application security structure is defined independently from the application itself. The logical security structure is stored in deployment descriptors of the application. For servlets, WebSphere Application Server allows you to protect the resources on the method level. For example, the POST method of a servlet can be part of a different security constraint than the GET method. The full list of predefined methods that can be secured as follows:

- GET
- POST
- PUT
- DELETE
- HEAD
- OPTION
- TRACE

Using method level security constraints for servlets, you might want to separate dynamic content that all the users can view from the administrative functions that only privileged users are allowed to access. In WebSphere Application Server, this is done using different security constraints for the different servlet methods.

17.2.3 Authorization

Authorization is the process of checking wether a given user has the privileges necessary to get access to the requested resource. WebSphere Application Server V6 supports authorization based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization. JACC enables third-party security providers to manage authorization in the application server. The default JACC provider that is provided by WebSphere Application Server uses the Tivoli Access Manager as the authorization provider.

When security is enabled in WebSphere Application Server V6, the default authorization is used unless a JACC provider is specified. The default authorization does not require special setup, and the default authorization engine makes all of the authorization decisions. However, if a JACC provider is configured and setup to be is used by WebSphere Application Server, the EJBs and Web authorization decisions are then delegated to the JACC provider.

When a JACC provider is used for authorization in WebSphere Application Server, the J2EE application-based authorization decisions are delegated to the provider per the JACC specification. However, all administrative security authorization decisions are made by the WebSphere Application Server default authorization engine. The JACC provider is not called to make the authorization decisions for administrative security.

When a protected J2EE resource is accessed, the authorization decision to give access to the principal is the same whether using the default authorization engine or a JACC provider. Both of the authorization models satisfy the J2EE specification, so there should be no differences in function. Choose a JACC provider only when you want to work with an external security provider such as the Tivoli Access Manager. In this instance, the security provider must support the JACC specification and be set up to work with the WebSphere Application Server. Setting up and configuring a JACC provider requires additional configuration steps, depending on the provider. Unless you have an external security provider that you can use with WebSphere Application Server, use the default authorization.

For additional details about JACC, refer to 17.4.6, “Java Authorization Contract for Containers” on page 502.

17.3 J2EE security

The notion of security includes several aspects. Identification is the process by which users or programs that communicate with each other provide identifying information. Authentication is the process used to validate the identity information provided by the communication partner. Authorization is the process by which a program determines whether a given identity is permitted to access a resource, such as a file or application component. The J2EE specifications describe the concepts to be used for these processes.

17.3.1 J2EE application

The J2EE specification defines the building blocks and elements of a J2EE application that build an enterprise application. The specification also provides details about security related to the different elements. The J2EE application consists of multiple modules and components; these elements are in connection with each other, and they communicate via certain protocols. A typical J2EE application consists of an application client tier, a Web tier, and EJB tier. When designing a security solution, you need to be aware of the connections between each of the modules. Figure 17-3 shows the components of a J2EE application.

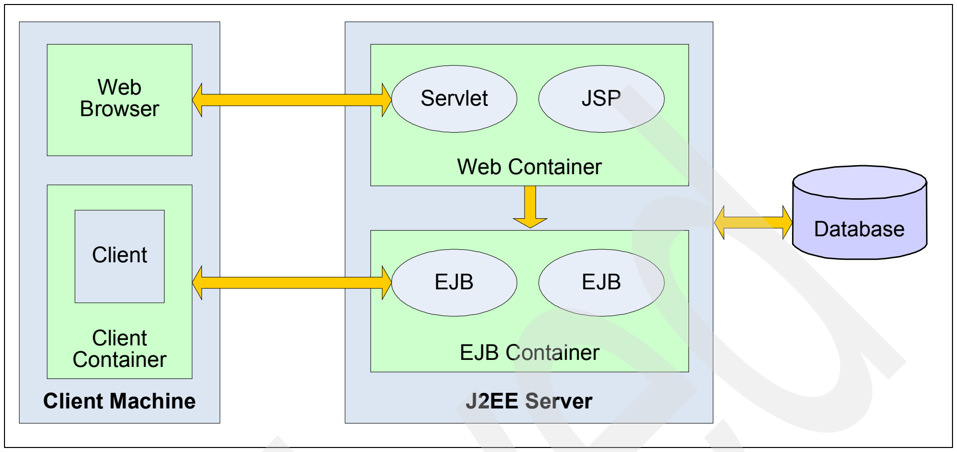


Figure 17-3   J2EE application components

For example, a user using a Web browser could access a JSP or a servlet, which is a protected resource. In this case, the Web container needs to check if the users is authenticated and has the required authorization to view the JSP or servlet. Similarly, a thick client could also access an EJB. When you plan for security, you need to consider the security for every module.

17.3.2 Security roles

A security role is a logical grouping of users that are defined by the application assembler. Because at development time, it is not possible to know all the users that are going to be using the application, security roles provide the developers a mechanism whereby the security policies for an application can be defined by creating named sets of users (for example managers, customers, employees, and so on) which have access to secure resources and methods. At application assembly time these users are just place holders. These sets of users or roles are mapped to real users or groups at deployment time. Figure 17-4 on page 497 shows an example of how roles could be mapped to users.

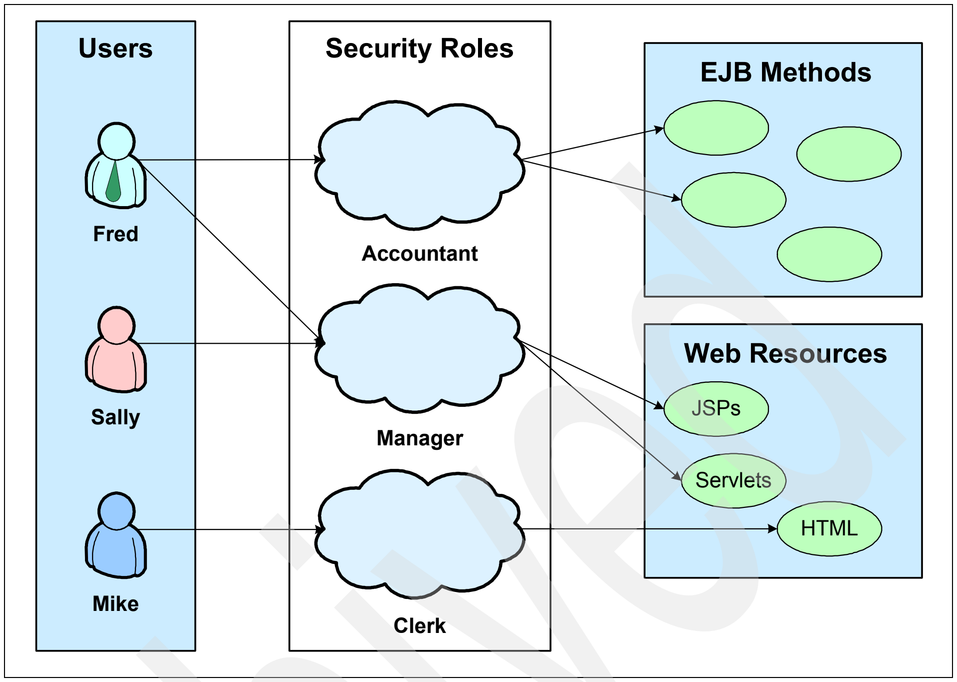


Figure 17-4 User role mapping

This two-phase approach to security gives a great deal of flexibility as deployers and administrators have a great control over how their users are mapped to the various security roles.

17.4 Programmatic security

J2EE containers enforce security in two ways:

- Declarative security
- Programmatic security

Most of the security is decided at application assembly or during development. This security configured during assembly time is called *declarative security*, because most of the security is defined or declared in the deployment descriptors. A deployment descriptor is an XML file which includes a representation of an application's security requirements, including the application's security roles, access control, and authentication requirements.

There are several descriptor files that are used for security role mappings. These files can be created using:

- IBM Rational Application Developer
- The Eclipse based Assembly tool that is shipped with WebSphere Application Server V6.

Programmatic security comes in handy when the application server provided security infrastructure cannot supply all the functions that are needed for the application. Using the Java APIs for security can be the way to implement security for the whole application without using the application server security functions at all. Programmatic security also gives you the option to implement dynamic security rules for your applications.

Generally, the developer does not have to code for security because WebSphere Application Server V6 provides a very robust security infrastructure which is transparent to the developer. However, there are cases where the security provided is not sufficient and the developer wants greater control over what the user has access to. For such cases, there are a few security APIs, which the developers can implement.

17.4.1 Servlet security methods

The servlet specification provides the following methods that allow programmatic access to the caller’s security information through the HttpServletRequest interface.

- String getRemoteUser()

  This method returns the user name the client used for authentication. This method returns a null if no user has authenticated.

- Boolean isUserInRole(String rolename)

  This method returns true if the remote user is granted the specified security role. If no user is authenticated or the remote user is not granted the specified authority, this method returns false.

- java.security.principal getUserPrincipal()

  This method allows the user to get the name of the current caller. In order to get the name of the user, you need to call the method getName() on the java.security.Principal object. This method returns a null if no user has authenticated.

17.4.2 EJB security methods

The EJB Specification provides two methods that can be called from the .EJBContext.

- java.security.principal getCallerPrincipal()

  The getCallerPrincipal method allows the developer to get the name of the current caller. To get the login name, you need to call the method getName() on the principal object. If the caller is not authenticated, it returns a principal containing UNAUTHENTICATED name.

- Boolean isCallerInRole(String rolename)

  This method returns true if the bean caller is granted the specified security role as specified by role name. If the caller is not granted the specified role, or if the caller is not authenticated, it returns false. If the specified role is granted Everyone access, it always returns true.

  When the isCallerInRole() method is used security-role-ref element needs to be declared in the deployment descriptor with a role-name sub-element containing the role name passed to this method. Because actual roles are created during the assembly stage of the application, a logical role can be used as the role name. During assembly, assembler creates a role-link sub-element to link the role-name to the actual role. Development tools such as Rational Web Developer can create security-role-ref elements. You also can create the security-role-ref element during the assembly stage using an assembly tool.

17.4.3 Custom user registry

A custom registry is a customer implemented registry that instruments the UserRegistry Java interface provided with WebSphere Application Server V6. The concept of custom user registry provides great flexibility to a customer environment where some form of user registry other than LDAP or Local operating system (LocalOS) is used.

By default, WebSphere Application Server V6 supports the following:

- LocalOS-based registry
- LDAP-based registry

However, there are cases where none of the options above are feasible and the user registry could be some other sort of custom registry like a database. For such cases WebSphere Application Server provides a service provider interface (SPI) that you can implement to interact with your current user registry. This SPI is the UserRegistry interface. This interface has a set of methods that need to be implemented for the product security to interact with your user registry for all security related tasks. The LocalOS and LDAP registry implementations that are shipped with the product also implement this same interface.

The UserRegistry interface is a collection of methods that are required to authenticate individual users using either password or certificates and to collect information about the user authorization purposes. This interface also includes methods that obtain user and group information so that they can be given access to resources. When implementing the methods in the interface, you must decide how to map the information that is manipulated by the UserRegistry interface to the information in your registry.

For further details and examples on custom user registry, refer to the WebSphere Application Server Information Center at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

17.4.4 Java Authentication and Authorization Service

The Java Authentication and Authorization Service (JAAS) extends the Java 2 security architecture with additional support to authenticate and enforce access control with principals and users. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization or principal-based authorization. WebSphere Application Server fully supports the JAAS architecture and extends the access control architecture to support role-based authorization for J2EE resources including servlets, JSP files, and EJB components.

JAAS authentication is performed in a pluggable fashion. This permits Java applications to remain independent from underlying authentication technologies. Therefore new or updated authentication technologies can be plugged under an application without requiring modifications to the application itself. Applications enable the authentication process by instantiating a LoginContext object, which in turn references a Configuration to determine the authentication technology, or LoginModule, to be used in performing the authentication. A typical LoginModule can prompt for and verify a user name and password.

A typical JAAS secured application has two parts.

- The main application that handles the login procedure and runs the secured code under the authenticated subject.
- The action that is invoked from the main application under a specific subject.

When using JAAS to authenticate a user, a subject is created to represent the authenticated user. A subject is comprised of a set of principals, where each principal represents an identity for that user. You can grant permissions in the policy to specific principals. After the user is authenticated, the application can associate the subject with the current access control context. For each subsequent security-checked operation, the Java run time automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject associated with the access control context contains the designated principal only.

Associate a subject with the current access control context by calling the static doAs method from the subject class, passing it an authenticated subject and java.security.PrivilegedAction or java.security.PrivilegedExceptionAction. The doAs method associates the provided subject with the current access control context and then invokes the run method from the action. The run method implementation contains all the code that ran as the specified subject. The action runs as the specified subject.

In the J2EE programming model, when invoking the EJB method from an enterprise bean or servlet, the method runs under the user identity that is determined by the run-as setting. The J2EE Version 1.4 Specification does not indicate which user identity to use when invoking an enterprise bean from a Subject.doAs action block within either the EJB code or the servlet code. A logical extension is to use the proper identity specified in the subject when invoking the EJB method within the Subject doAs action block.

For further details about JAAS, refer to the WebSphere Application Server Information Center at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

17.4.5 Trust Association Interceptors

Web Clients can also authenticate by using a Trust Association Interceptor (TAI). Trust association enables the integration of WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the reverse proxy server.

Demand for such an integrated configuration has become more compelling, especially when a single product cannot meet all of the customer needs or when migration is not a viable solution. In this configuration, WebSphere Application Server is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes the HTTP request to the WebSphere Application Server that includes the credentials of the authenticated user. WebSphere Application Server then uses these credentials to authorize the request.

Trust Association Model

The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every Web request that is dispatched to WebSphere Application Server. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

Running in trust association mode does not prohibit WebSphere Application Server from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust. It is possible, however, to configure WebSphere Application Server to strictly require that all HTTP requests go through a reverse proxy server. In this case, all requests that do not come from a proxy server are immediately denied by WebSphere Application Server.

Trust association interceptor interface

The intent of the trust association interceptor interface is to have reverse proxy security servers (RPSS) as the exposed entry points to perform authentication and coarse-grained authorization, while the WebSphere Application Server enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, the distributed environment of a company consists of Web servers, application servers, legacy systems, and one or more RPSS, such as the Tivoli WebSEAL product. Such reverse proxy servers, front-end security servers, or security plug-ins registered within Web servers, guard the HTTP access requests to the Web servers and the application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization, and request routing to the target application server.

17.4.6 Java Authorization Contract for Containers

JACC is a new specification introduced in J2EE 1.4 through the Java Specifications Request JSR-115. This specification defines a contract between J2EE containers and authorization providers. The contract enables third-party authorization providers to plug into J2EE 1.4 application servers (such as WebSphere Application Server) to make the authorization decisions when a J2EE resource is accessed. The access decisions are made through the standard java.security.Policy object.

In WebSphere Application Server, two authorization contracts are supported using both a native and a third-party JACC provider implementation. The default (out-of-box) solution is the WebSphere Application Server default J2EE role based authorization implementation, which does not implement the JACC Policy provider interface. The JACC specification does not specify how to handle the authorization table (user or group to role) information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. It does not require the container to provide the authorization table information in the binding file to the provider.

The JACC specification does not specify how to handle the authorization table information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. It does not require the container to provide the authorization table information in the binding file to the provider.

WebSphere Application Server supports authorization based on the JACC specification in addition to the default authorization. It enables third-party security providers to manage authorization in the application server. The default JACC provider that is provided by WebSphere Application Server uses the Tivoli Access Manager as the authorization provider.

WebSphere Application Server provides two role configuration interfaces (RoleConfigurationFactory and RoleConfiguration) to help the provider obtain information from the binding file, as well as an initialization interface (InitializeJACCProvider). The implementation of these interfaces is optional.

The JACC provider in WebSphere Application Server is implemented by both the client and the server pieces of the Tivoli Access Manager server. The client piece of Tivoli Access Manager is embedded in WebSphere Application Server. The server piece is located on a separate installable CD that is shipped as part of the WebSphere Application Server Network Deployment V6 package.

The JACC provider is not an out-of-box solution. You must configure WebSphere Application Server to use the JACC provider.

JACC access decisions

When security is enabled and an enterprise bean or Web resource is accessed, the EJB container or Web container calls the security run time to make an authorization decision on whether to permit access. When using an external provider, the access decision is delegated to that provider. According to the JACC specification, the appropriate permission object is created, the appropriate policy context handlers are registered, and the appropriate policy context identifier (contextID) is set. A call is made to the java.security.Policy object method implemented by the provider to make the access decision.

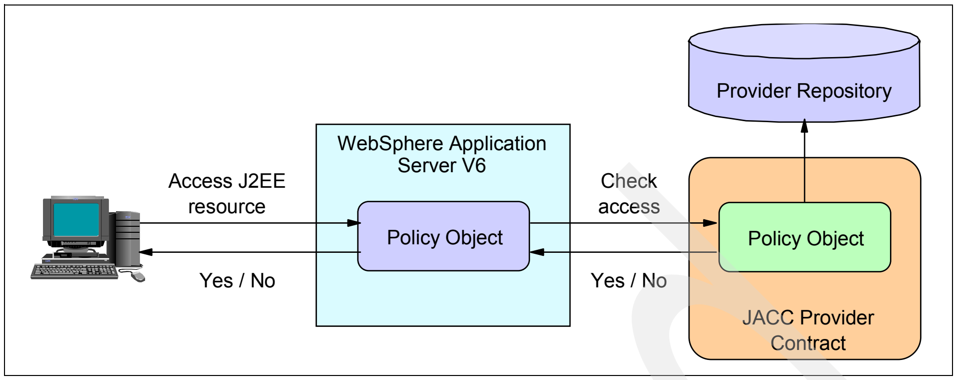


Figure 17-5 JACC provider architecture

Dynamic module updates in JACC

WebSphere Application Server supports dynamic updates to Web modules under certain conditions. If a Web module is updated, deleted or added to an application, only that module is stopped or started as appropriate. The other existing modules in the application are not impacted, and the application itself is not stopped and then restarted.

When any security policies are modified in the Web modules, the application is stopped and then restarted when using the default authorization engine. When using the JACC-based authorization, the behavior depends on the functionality that a provider supports. If a provider can handle dynamic changes to the Web modules, then only the Web modules are impacted. Otherwise, the entire application is stopped and restarted for the new changes in the Web modules to take effect.

A provider can indicate if they support the dynamic updates by configuring the supports dynamic module updates option in the JACC configuration model. This option can be enabled or disabled using the administrative console or by scripting. It is expected that most providers store the policy information in their external repository, which makes it possible for them to support these dynamic updates. This option is enabled by default for most providers.

When the supports dynamic module updates option is enabled, if a Web module that contains security roles is dynamically added, modified, or deleted, only the specific Web modules are impacted and restarted. If the option is disabled, the entire application is restarted. When dynamic updates are performed, the security policy information of the modules impacted are propagated to the provider.

Mixed node environment and JACC

Authorization using JACC is a new feature in WebSphere Application Server V6. Previous versions of the WebSphere Application Server do not support this feature. Also, the JACC configuration is set up at the cell level and is applicable for all the nodes and servers in that cell.

If you are planning to use the JACC-based authorization the cell should only contain WebSphere Application Server V6 nodes. This means that a mixed node environment containing a set of V5 or later nodes in a WebSphere Application Server V6 cell is not supported.

18

Planning for migration

This chapter discusses the planning for migration, upgrades, coexistence, and interoperability. The are many complex issues concerning migration, and all need to be carefully understood. WebSphere Application Server V6 provides new migration tools to simplify the transition process. The chapter contains the following sections:

- Migration overview
- Migrating product configurations
- Migrating WebSphere applications
- Migrating administrative scripts
- Migrating Web server configurations
- Migration concerns
- Migration troubleshooting
- Summary

18.1 Migration overview

WebSphere Application Server includes migration tools that provide migration functionality. The migration wizard is new in this version and provides a graphical user interface for the migration tools. The migration tools can be used through the migration wizard or can be executed directly from a command line. The migration tools migrate applications and configuration information to the new version of WebSphere Application Server.

It is also possible to run more than one version of WebSphere Application Server at the same time if there are no conflict with the ports assigned. In WebSphere Application Server V6, these ports are specified when creating a profile.

Migrating and coexisting have roughly opposite goals. The goal of migration is to reconstruct your current environment on a nearly identical WebSphere Application Server V6 environment, this is including applications, configuration settings, universal resource identifier (URI) descriptors, Web server context roots, and so forth. The goal of coexistence is to create a new environment that is not in conflict with an earlier version, so far as port settings are concerned. This allow both environments to start and run at the same time.

18.1.1 General migration considerations

Although every migration scenario is different, there are some general considerations to take into account. The first is identifying what is to be migrated. After you have determined what to migrate, it is important to develop a plan to perform the migrations. The plan should also consider if there will be any coexistence or interoperability.

Migrate product and Web server configurations

The first step is to migrate the WebSphere product to V6. You need to install WebSphere Application Server V6 and then migrate the configuration files to the new WebSphere Application Server V6 environment.

The following is a brief overview of the product and server migrations:

- Obtain an overview of migration and coexistence options. Find out about supported migration paths, migration tools, the goals of migration and coexistence, and more.
- Review the software and hardware prerequisites. Learn which product configurations are supported and which need to be applied to the environment before and after the migration.

- Determine what configuration information needs to change. Learn what changes during migration, which involves migrating a single instance to another single instance on the same machine or a separate machine.
- Locate migration tools. Become acquainted with the tools that the product provides for automated migration.
- Migrate Web server plug-in configurations. Migrate the configurations used by the Application Server to communicate with Web server.

Migrate your applications

After the product and the Web server have been migrated to WebSphere Application Server V6, it is time to migrate the applications. The following is a brief overview for migrating specific application components, their deployment descriptors, and their administrative configurations:

- Identify deprecated features that you might be using. Review a summary of deprecated features in WebSphere Application Server V6.
- Review the software and hardware prerequisites. Learn which product configurations are supported.
- Learn about WebSphere applications. Investigate the technologies used in and by applications that you deploy on the Application Server.

Coexistence

Coexistence, as it applies to WebSphere Application Server products, is the ability of multiple installations of WebSphere Application Server to run on the same machine at the same time. Multiple installations include multiple versions and multiple instances of one version. Coexistence also implies various combinations of Web server interaction.

The following is a brief overview for coexistence scenarios:

- Review supported coexistence scenarios. All combinations of V4.x, V5.x, and V6 products can coexist as long as there are no port conflicts. There are some coexistence limitations for V5 products that have the embedded messaging feature installed. The installation wizard looks for certain existing installations to determine if it should prompt for coexistence information.

  For further details, refer to the WebSphere Application Server V6 Information Center at:

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`
- Configure ports for coexistence. Identify port numbers assigned in the different versions of WebSphere Application Server.
- Obtain valid port settings for coexistence. V6 autonomically detects free ports among profiles installed from the same core product files. The port

mechanism does not detect ports in use by previous versions or by other products. You must verify that you are using ports that do not conflict with other products and programs running on that node. Select free ports on the ports selection panel in the Profile creation wizard.

Interoperability

Interoperability is exchanging data between two co-existing product installations. The following is a brief overview of the interoperability process:

- Interoperate across product versions. WebSphere Application Server V6 is generally interoperable with some previous versions. However, there are specific requirements to address for each version. Certain changes are required to support interoperability between versions.
- Configure ports for coexistence or interoperability. Identify port numbers assigned in the different versions of WebSphere Application Server.

18.1.2 Deprecated features in WebSphere Application Server V6

While planning for migrating to WebSphere Application Server V6, it is important to understand which features are no longer valid in the new version and how to handle them. Refer to 3.7, “Deprecated features in this version” on page 69 for further details. Additionally visit the WebSphere Application Server V6 Information Center for the most up-to-date information.

http://www.ibm.com/software/webservers/appserv/infocenter.html

The deprecated features can be summarized into a number of different categories for planning consideration.

Application programming model features

These are the programming features needed to assemble and deploy applications. You need to understand how the deprecated features impact the application and develop a migration plan to handle them.

Application services features

Application services are the components that provide the inter-connectivity needed to run an enterprise application. Common application services are UDDI, J2EE connectors, data sources, Web services, and deployment descriptors. These components commonly change with J2EE versions and as WebSphere changes to accommodate, the application also has to change.

Security features

Another area that changes frequently with new J2EE features is security. As new security standards emerge, old security features must be migrated to meet these new standards. WSS (Web Service Security) is implemented in WebSphere Application Server V6 and older security standards, such as SOAP, need to be migrated to meet the new standards.

System administration features

Various methods for configuration and installation have changed under WebSphere Application Server V6 and these changes must be considered in your migration plan. The basic administrative architecture differences between your current version and WebSphere Application Server V6 needs to be studied to eliminate troublesome areas. The new WebSphere Application Server V6 cell configuration features has deprecated many configuration and installation features of earlier versions.

Performance features

New performance monitoring features in WebSphere Application Server V6 has replaced the data gathering features of earlier versions. The new PMI architecture in WebSphere Application Server V6 allows the user to monitor the overall health of the system. Old PMI are no longer compatible with the new PMI architecture and, therefore, must be migrated.

18.1.3 New installation features

A major installation improvement is the ability to install one copy of the core files on a machine and then use profiles to define multiple Application Server runtime environments. Each runtime environment can have its own administrative interfaces that share the core files. This has many positive ramifications for preparing, installing, maintaining, and removing installations, including a decreased footprint.

Another change is the introduction of separate installation routines for the Application Server product, the Web server, the Web server plug-ins, and the clients so that you can install only what you need on a particular machine. The ability to incrementally upgrade nodes in a cell is another new feature that allows a more granular installation procedure to simplify installation and improve installation reliability. There is also a redesigned Launchpad to plan and start installation. The Launchpad also provides product solution roadmap diagrams to let you decide on the fly what to install where.

18.1.4 Migration tools

All of the migration tools are in the install_root/bin directory after installation. It is important to use the migration tools for the version of WebSphere Application Server that you are installing because the tools change over time. If you use migration tools from an earlier release, you might encounter issues with the migration.

The WebSphere Application Server V6 migration tools are:

- Migration wizard

  The migration wizard is new in WebSphere Application Server V6. The wizard is the GUI to the WebSphere Application Server V6 migration tools which are the WASPreUpgrade command and the WASPostUpgrade command. Before using the migration wizard, you must have access to the existing, previous, version of WebSphere Application Server and you must also have a valid V6 profile.

- clientUpgrade.sh (and clientUpgrade.bat)

  Upgrades the client application to a new release level.

- convertScriptCompatibility.sh (and convertScriptCompatibility.bat)

  Used by administrators to convert their configuration from a mode that supports backward compatibility of V5.x administration scripts to a mode that is fully V6.

- WASPreUpgrade.sh (and WASPreUpgrade.bat)

  Saves the applications and configuration data from a previous installation of WebSphere Application Server to a backup directory. The WASPostUpgrade script restores the configuration data from the backup directory to the new installation. The migration wizard calls the WASPreUpgrade.sh script during migration. You can also run these scripts from the command line to perform a manual migration.

- WASPostUpgrade.sh (and WASPostUpgrade.bat)

  Restores the configuration data from a previous release. WASPostUpgrade reads the data from the backup directory where the WASPreUpgrade script stored the data. The migration wizard calls the WASPostUpgrade.sh script during migration.

18.1.5 Granular installation

A new feature of WebSphere Application Server V6 is the incremental cell upgrade. This new version of the Deployment Manager can manage both V6 and V5.x nodes. This allows the cell to be upgraded to a new release one node at a time, with minimal impact to the applications that are running within the cell. When upgrading to V6, there is no need to export the cell configuration and recreate it in a new cell. However, there are limitations and restrictions on the node, server, cluster, and application operations in a mixed node environment.

Migrate the Network Deployment node from V5.x to V6 first, then migrate the base nodes that comprise the cell. The Network Deployment node must always be at the highest release and fix level within a cell, to allow to manage all nodes in the cell. If your cluster members in a mixed release cell include V6 clusters and older versions from 5.0 through 5.1.0, there is an Object Request Broker (ORB) custom property of which you should be aware. This does not apply if you are running a mixed release cell with V6 and V5.1.1 servers.

In appropriate cases, the migration program automatically sets a custom property for the ORB on the Deployment Manager V6, node agent, and servers in the cluster. After migrating the entire cell to V6, you can reset the property for a performance improvement.

18.1.6 Client installation wizard

Running J2EE and thin application clients that communicate with WebSphere Application Server requires that elements of the Application Server are installed on the machine on which the client runs. However, if the system does not have the Application Server installed, you can install Application Clients, which provide a stand-alone client run-time environment for your client applications. WebSphere Application Server V6 can be installed on a machine with a previous version of Application Clients. However, you cannot install a V6 Application Client on top of a previous version of the Application Client.

When you install application clients, the current working directory must be the directory where the installer binary program is located. This placement is important because the resolution of the class files location is done in the current working directory. Failing to use the correct working directory can cause ISMP errors that abort the installation. The installation wizard does not upgrade or remove any previous Application Client installation automatically.

18.2 Migrating product configurations

Administrative configurations can be migrated with the migration wizard or with the command line migration tools. If you used an earlier version of WebSphere Application Server, the system administrator might have fine-tuned various application and server settings for your environment. It is important to have a plan for migrating these settings with maximum efficiency and minimal loss. The migration tools in WebSphere Application Server V6 support migration from the following versions of WebSphere Application Server: V4.0.x, V5.0.x, V5.1.x.

18.2.1 Incremental upgrade

Incremental migrations of earlier version nodes can be performed by calling the migration tools multiple times, each time specifying a different configuration file. Various reasons exist for having multiple configuration files. Whatever the reason is, migrating one configuration file at a time lets you test applications incrementally before continuing to the next configuration file.

Before using the migration tools, consult the WebSphere Application Server V6 Release notes document to understand what fixes you must apply to earlier versions. Applying fixes to an earlier version might also apply fixes to files that have a role in the migration. Apply any fixes to ensure the most effective migration of configurations and applications possible.

A V5.x node cannot be added to a V6 cell, instead that node should be migrated to V6 first and then added to the cell. There are restrictions on creating clusters and cluster members in a mixed version scenario.

The following are supported:

- Existing V5.x application servers and JMS servers continues to work.
- Removing a server at any version level.
- Adding a new V6 member to a cluster the mixed cluster already contains a V6 member through migration of one of its members from V5.x to V6.

The following are not supported:

- Creating new V5.x application servers and JMS servers.
- Converting a V5.x server to a cluster.
- Adding a new V5.x member to a cluster that already contains a 5.x server. To add a new member, all cluster members must first be upgraded.

18.2.2 Configuration mapping

Many migration scenarios are possible. The WebSphere Application Server V6 migration tools map objects and attributes to the V6 environment when you restore a configuration from a previous version. Understanding what objects and attributes change during the migration process beforehand eases the transition to the new version. The WebSphere Application Server V6 migration alters the configuration mapping of the following:

- Bootstrap port

  Migration maps a default bootstrap NameServer port setting, 900, from V4.0.x Advanced Edition or V5.x to the V6 NameServer default, 2809.

- Command line parameters

  The migration tools convert appropriate command line parameters to Java virtual machine (JVM) settings in the server process definition.

- Cluster members

  V4.0.x server groups are converted to V6 clusters. When migrating a server group, all servers in the group are assigned to the HTTP transport port number of the server group, regardless of whether or not they had a unique port number.

- Default server

  The name of the default server in V6 is server1.

- GenericServer

  Introduced in V5.1.x, a GenericServer was an APPLICATION_SERVER fitted to manage external resources. In V6, it has its own type, called GENERIC_SERVER.

- Java DataBase Connectivity

  V6 significantly redefines JDBC and data source objects. The migration tools map V4.0.x data sources to V6 data sources, using predecessor settings as input variables.

- jmsserver

  Changed from type MESSAGE_BROKER to type APPLICATION_SERVER. Any queues or topics that it owned have been migrated into the new WebSphere Platform messaging framework.

- Migration of a V5.x node to a V6 node

  You can migrate a V5.x node that belongs to a cell without removing the node from the cell.

- Name bindings

  WebSphere Application Server V6 has a new naming structure. All references, such as Enterprise JavaBeans (EJB) references that were valid in previous versions no longer work in V6.

- Node name

  The tool identifies node names in the configuration files that it is migrating and selects any node names in a configuration file that match the long network name or short network name of the migrating machine.

- PageList servlet

  Direct use of the servlet has been deprecated. All references are updated to the servlet configuration supported in V6.

- Policy file migration from V5.x to V6

  WebSphere Application Server V6 migrates all the policy files that are installed with V5.x by merging settings into the V6 policy files.
- Properties directory and the lib/app directory

  Migration copies files from prior version directories into the V6 configuration.
- Property file migration from V4.0.x

  V6 migration migrates property files from V4.0.x if these files are also present in WebSphere Application Server V6.
- Property file migration from V5.x to V6

  WebSphere Application Server V6 migrates all the property files that are installed with V5.x by merging settings into the V6 property files.
- Resource Adapter Archive (RAR) referenced by J2C resources

  RAR that are referenced by J2C resources are migrated if those RARs are in the old WebSphere Application Server installation.
- Security

  Java 2 Security is enabled by default in WebSphere Application Server V6.
- Server groups

  V4.0.x server groups are redefined in V6 as clusters.
- Servlet package name changes - In WebSphere Application Server V6 the servlets are in the com.ibm.ws.webcontainer.servlet class.
- Transport ports

  The migration tools migrate all ports and warn of any conflicts.
- Web modules

  The specification level of the Java 2 Platform, Enterprise Edition (J2EE) that WebSphere Application Server V6 implements requires behavior changes in the Web container for setting the content type.
- Java heap size for migrating EAR files

  When migrating all V5.x EAR files to V6 using the **`wsadmin`** tool, the WASPostUpgrade tool uses the default maximum Java heap size value of 64MB to install the EAR files.

18.2.3 Deployment manager migration

Before the Deployment Manager can be migrated to V6, the WebSphere Application Server and configurations must be upgraded to V6. The Deployment Manager must always be at the highest release and fix level within a cell to allow it to manage all nodes in the cell. In WebSphere Application Server V6, the Deployment Manager has the capability to manage both V6 and V5.x nodes. This allows a cell to be upgraded to a new release one node at a time, with minimal impact to the applications that are running within the cell.

18.2.4 Migrating WebSphere PME

The migration of Programming Model Extensions (PME) services from WebSphere Application Server Enterprise Edition V4.0.x and V5.0.x and from WebSphere Business Integration Server Foundation V5.1.x to WebSphere Application Server V6 is handled on individual basis. For PME services that are not supported in WebSphere Application Server V6, all configuration information is removed. For PME services that are supported in WebSphere Application Server V6, the configuration from the previous release environment overwrites the values in the new release.

Validating PMEs

As part of application installation, both during migration and outside of migration, applications are validated to ensure that they only use resources for services that are supported by WebSphere Application Server V6. Any application that uses a resource for a service that is not supported by WebSphere Application Server V6 will not work correctly, and an error will be issued indicating that the application cannot be installed.

Running a mixed-node environment

When running in a mixed-node environment (such as a WebSphere Application Server V6 Deployment Manager managing WebSphere Business Integration Server Foundation V5.1.x nodes), the first syncNode performed by the system downloads a configuration from the WebSphere Application Server V6 Deployment Manager to the WebSphere Business Integration Server Foundation V5.1.x nodes. Therefore, any PME service that is not supported by WebSphere Application Server V6 is rendered inoperable on the WebSphere Business Integration Server Foundation V5.1.x node.

18.3 Migrating WebSphere applications

When the product and configurations have been migrated to WebSphere Application Server V6, then the applications can be migrated to the new version. If any deprecated features exist, a plan must be developed to handle the deprecated features. In most cases, IBM supplies a recommended method for a deprecated feature in the WebSphere Application Server V6. Also, only certain product configurations are supported and if the current configuration is not supported a comprehensive review is necessary to migrate the application. And finally, it is important to understand the new technologies in WebSphere Application Server V6 to fully take advantage of them.

For further details refer to the WebSphere Application Server V6 Information Center at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

18.3.1 Incremental upgrade

As mentioned earlier, nodes can be upgraded one node at a time and different version nodes can coexist in one cell. Therefore, various application versions can be supported in one runtime environment easily. When upgrading to V6 there is no need to export the cell configuration and recreate it in a new cell. However, there are limitations and restrictions on the node, server, cluster, and application operations in a mixed node environment.

If your cluster members in a mixed version cell include V6 clusters and older versions from V5.0 through V5.1.0, there is an Object Request Broker (ORB) custom property of which you should be aware. If running a mixed version cell with V6.0 and V5.1.1 servers, you can disregard this note. In appropriate cases, the migration program automatically sets a custom property for the ORB on the WebSphere Application Server V6 Deployment Manager, node agent, and servers in the cluster. After migrating the entire cell to V6, you can reset the property for a performance improvement.

18.3.2 Upgrade Web server plug-in

WebSphere Application Server products supply a unique binary plug-in module for each supported Web server. The plug-in installation wizard installs required files and configures the Web server and the application server to allow communication between the servers.

You must install a supported Web server before you can install a plug-in for the Web server. If the WebSphere Application Server product is not installed, you can still install the plug-in.

IBM HTTP Server V6 can coexist with earlier versions, or you can upgrade earlier versions to V6. Upgrading relieves you from having to uninstall and reinstall the HTTP server. Install IBM HTTP Server V6 into the same directory structure as the earlier version to upgrade that version. If you install the IBM HTTP Server into a different directory, IBM HTTP Server V6 coexists with the previous version.

The plug-ins installation wizard installs the plug-in module, configures the Web server for communicating with the application server, and creates a Web server configuration definition in the application server, if possible.

Some topologies, such as the Web server on one machine and the application server on another machine, prevent the plug-ins installation wizard from creating the Web server definition in the application server configuration on the remote machine. In such a case, the plug-ins installation wizard creates a script that you can copy to the application server machine. Run the script to create the Web server configuration definition within the application server configuration.

18.3.3 Replication domains

In WebSphere Application Server V6, data replication domains replace multi-broker replication domains that were available for replication in earlier versions. Any replication domains that are created with a previous version of WebSphere Application Server might be multi-broker domains. Migrate any multi-broker domains to the new data replication domains. Although you can configure existing multi-broker domains with the current version of WebSphere Application Server, after you upgrade your Deployment Manager, you can create only data replication domains in the administrative console.

Multi-broker and data replication domains both perform the same function, which is to replicate data across the consumers in a replication domain. Configure all the instances of replication that need to communicate in the same replication domain.

After upgrading the Deployment Manager to the latest version of WebSphere Application Server, only data replication domains can be created. Any multi-broker domains that you created with a previous release of WebSphere Application Server are still functional, however, new multi-broker domains or replicators cannot be created. You can also configure the session manager with both types of replication domains to use topologies such as peer-to-peer and client/server to isolate the function of creating and storing replicas on separate application servers.

WebSphere Application Server V6 uses data replication domain objects that contain configuration information to configure the high availability framework on

WebSphere Application Server. The transport is no longer based on the JMS API. Therefore, no replicators and no JMS brokers exist. You do not have to perform the complex task of configuring local, remote, and alternate replicators.

Earlier versions of WebSphere Application Server encouraged the sharing of replication domains between different consumers, such as session manager and dynamic cache. WebSphere Application Server V6 encourages creating a separate replication domain for each consumer. For example, create one replication domain for session manager and another replication domain for dynamic cache. The only situation where you should configure one replication domain is when configuring session manager replication and stateful session bean failover. Using one replication domain in this case ensures that the backup state information of HTTP sessions and stateful session beans are on the same application servers.

18.4 Migrating administrative scripts

In many cases, scripts were created to automate many common administrative tasks. The WebSphere Application Server V6 migration tools provide a method to migrate administrative scripts from V4.X and V5.X to V6. The `wsadmin` tool is the scripting tool for WebSphere Application Server V6 and can adapt administrative scripts from earlier versions with forethought and planning.

18.4.1 Migrating from V4.X

Migrating scripts from V4.X to V6 is a bit more difficult then migrating from V5.X but can be accomplished with a little planning. The `wscp` tool was a part of the WebSphere Application Server V4.0 administration repository support. The repository no longer exists and the tools that manipulate it are no longer needed. You can use the V6 scripting client program, wsadmin, to do the same kinds of things `wscp` did, and more. You can use the Jacl and Jython scripting languages for scripts, but the elements specific to `wsadmin` are different from those available in `wscp`.

Automatic conversion of scripts between the two releases is difficult. In WebSphere Application Server V4.0, `wscp` commands are used for both configuration queries or updates, and operational commands. In WebSphere Application Server V6 a distinction is made between configurational and operational commands.

It is necessary to find the corresponding configuration for `wsadmin` V6 object type for each configuration command. Use the AdminConfig create, list, modify, remove, show, and showAttribute commands to perform the same type of operations in V6 that you performed in V4.0. Determine the V6 attribute names by using the online help commands of the AdminConfig object. Operational commands, such as start and stop server, can similarly be converted.

To convert application installation commands, use the AdminApp object install Interactive command to complete a successful installation. Then locate message **WASX7278I** in the `wsadmin.traceout log` file and use the data in the message to construct an installation command for your source. In V6 configuration changes are made to a temporary workspace. These changes are not committed to the WebSphere Application Server configuration until you invoke the **`save`** command on the AdminConfig object.

18.4.2 Migrating from V5.X

There are a few changes to be aware of that are required for your existing scripts when moving to WebSphere Application Server V6 from V5. In general, the administration model has changed very little and most administrative scripts will function properly. However, there are some changes required with V6.

During migration, the default is to migrate using compatibility mode. If this option is taken, then the old object types are migrated into the new configuration and existing scripts will run unchanged. The old HTTPTransport objects are migrated and mapped onto the new channel architecture. Existing scripts can modify these objects and will run unchanged.

Be aware of the implications of migrating JMS applications from the embedded messaging in WebSphere Application Server V5 to the default messaging provider in WebSphere Application Server V6. A new version of Jacl (1.3.1) is shipped with WebSphere Application Server V6. With this Jacl version, regexp command supports only Tcl 8.0 regexp command syntax. If your existing V5.x Jacl script uses regexp command syntax that is supported in Jacl 1.2.6 but not anymore in Jacl 1.3.1, you might not get a match anymore, or you might get a compile error for your regexp command. There is no work around for this regression problem. Jacl has indicated that this is by design and there is no simple patch to fix this design decision.

18.5 Migrating Web server configurations

IBM HTTP Server V6 can coexist with earlier versions, or you can upgrade earlier versions to V6. Upgrading relieves you from having to uninstall and reinstall the HTTP server. Install IBM HTTP Server V6 into the same directory structure as the earlier version to upgrade that version. If you install the IBM HTTP Server V6 into a different directory, IBM HTTP Server V6 coexists with the previous version.

By default, the administration server and the Web server use the same ports as the previous version, which causes a conflict. However, you can change the port assignments on the port assignment panel of the WebSphere Application Server installation wizard or the Profile creation wizard.

Migrate plug-ins to work with WebSphere Application Server V6. The Web server plug-in configuration file (plugin-cfg.xml) is associated with every Web server definition instead of one cell-wide plug-in configuration file. The settings in the generated plug-in configuration file are based in the list of applications that are deployed on the hosting Web server. Use the plug-ins installation wizard to configure the Web server. Instead of using the default plug-in configuration file location, specify the new location of the plug-in configuration file that was propagated in the previous step.

18.5.1 Plug-ins configuration wizard

After installing a supported Web server, you must install a binary plug-in module for the Web server. The plug-in module lets the Web server communicate with the Application Server. The plug-ins installation wizard installs the Web server plug-in and configures the Web server. The wizard also creates a Web server definition in the configuration of the application server.

The plug-in installation wizard uses five files to configure a plug-in for the Web server you select:

- Web server configuration file

  The Web server configuration file is installed as part of the Web server. The wizard must reconfigure the configuration file and add directives that identify file locations of two files: the binary plug-in file and the plugin-cfg.xml configuration file.

- The binary Web server plug-in file

  The binary plug-in file does not change. However, the configuration file for the binary plug-in is an XML file. The application server changes the configuration file whenever you change the configuration of deployed applications. The binary module reads the XML file to adjust settings and to locate deployed applications for the Web server.

- The plug-in configuration file, plugin-cfg.xml

  The plug-in configuration file is an XML file with settings that you can tune in the administrative console. The file lists all of the applications installed on the Web server definition. The binary module reads the XML file to adjust settings and to locate applications for the Web server. The Web server plug-in configuration service automatically regenerates the plugin-cfg.xml file whenever the configuration changes. The configuration service automatically propagates the plugin-cfg.xml file to an IBM HTTP Server machine. You must manually copy the file on other Web servers.

- Default plug-in configuration file, plugin-cfg.xml

  The default file is a placeholder that you must replace with the plugin-cfg.xml file from the Web server definition on the application server. The default file is a replica of the file that the Application Server creates for a default stand-alone application server that has the samples installed. After the Web server definition is created, the Web server plug-in configuration service within the application server creates the first plugin-cfg.xml file in the Web server definition on the application server machine. If you install an application, create a virtual host, or do anything that changes the configuration, you must propagate the updated plugin-cfg.xml file from the Application Server machine to the Web server machine to replace the default file.

- The configureWeb_server_name script

  The plug-in installation wizard creates the configureWeb_server_name script on the Web server machine. The script is created for remote installation scenarios only. Copy the script from the Web server machine to the remote application server machines. You do not have to copy the script on a local installation. Run the script to create a Web server definition in the configuration of the application server. Each stand-alone application server can have only one Web server definition. A managed node, on the other hand, can have multiple Web server definitions. The script creates a new Web server definition unless the Web server name is the same.

All of the default servlets and applications are included in the default file, including the WSsamples application and the SamplesGallery application. The default file lists all of the default applications and samples. The list can be inaccurate. If you performed a custom installation and did not install the samples, for example, the list is inaccurate. To serve an application that you developed with the Web server, propagate the real plugin-cfg.xml file.

18.6 Migration concerns

After the product, configurations, and applications have been migrated, you need to consider the day-to-day operations of the environment. In many instances, V6 is not the only version of WebSphere in the environment. The co-existence or interoperability of V6 with any other version needs careful planning. Depending on from which version the migration occurred, there can be some post-migration configuration necessary. All of these factors must be considered before the migration is complete.

18.6.1 Coexistence

Coexistence is the process of running a new version of a WebSphere Application Server product on the same machine at the same time as you run an earlier version, or running two different installations of the same version of a WebSphere Application Server product on the same machine at the same time.

Migrating and coexisting have roughly opposite goals. The goal of migration is to reconstruct your earlier version in a nearly identical V6 environment, including applications, configuration settings, URI descriptors, and Web server context roots. The migration wizard uses the migration tools to migrate the previous configuration and applications. The goal of coexistence is to create an environment that is not in conflict with an earlier version, so far as port settings are concerned. This allows both nodes to start and run at the same time.

18.6.2 Interoperability

WebSphere Application Server V6 is generally interoperable with WebSphere Application Server V4.0.x and V5.x. However, there are specific requirements to address for each version. In general, IBM recommends that you apply the latest fix level to support interoperability.

Fixes are available on the support site for WebSphere Application Server products at:

`http://www.ibm.com/software/webservers/appserv/support`

Interoperability guidelines

A number of guidelines can be used to aid in achieving interoperability with WebSphere Application Server V6:

- Set the following JVM properties:
  - com.ibm.ejs.jts.jts.ControlSet.nativeOnly=false
  - com.ibm.ejs.jts.jts.ControlSet.interoperabilityOnly=true

  Always apply this guideline to V4.0.x. For V5.0.2, apply this guideline in addition to applying interim fixes (or moving to V5.0.2.8).
- Make required naming changes to support V4.0.x client access to V6 enterprise beans. There are several ways to make it work, such as:
  - Updating the namebindings.xml file if you do not use the V6 migration tools, but have installed V4.0.x applications on WebSphere Application Server V6. To allow V4.0.x client access to the applications, add additional information to the bind information in the V6 namespace to make the old JNDI names work.

Add the information to the namebindings.xml configuration file at the cell level using the administrative console. After federating an application server node into a Deployment Manager cell, you cannot use the migration tools. To use these tools again, remove the node from the cell, use the tools, and add the node to the cell again.

– Calling WebSphere Application Server V6 enterprise beans directly using the naming path that includes the server on which the enterprise beans are running, such as cell/node/server1/some/ejb/name, for example.

– Using the V4.0.x client java:/comp location to find WebSphere Application Server V6 enterprise beans.

- Be aware of administrative console limitations. You cannot use the administrative interfaces for WebSphere Application Server V6 to administer a V4.0.x administrative server. Likewise, you cannot use a WebSphere Application Server V4.0.x administrative console to administer a V6 environment.

  If you use the administrative console on a remote machine to administer WebSphere Application Server V4.0.x domains, migrating any of the nodes or domains to WebSphere Application Server V6 renders the remote administrative console ineffective for administering any WebSphere Application Server V6 environment.

- Use Secure Sockets Layer Version 3 (SSL v3) for secure connections when inter-operating . You cannot use Common Secure Interoperability Version 2 (CSIv2) for interoperability, because V4.0.x does not support CSIv2.

- Be aware of limitations when calling WorkLoad Management (WLM)-enabled enterprise beans. Some clients cannot call WLM-enabled enterprise beans on remote clusters when there is a local WLM-enabled enterprise bean of the same name.

  If there is a cluster local to the client with the same enterprise bean as the remote cluster that the client is trying to call, the client ends up talking to the local cluster.

- Be aware of the level of WebSphere Application Server in which each function you use is supported. Applications that you intend to be interoperable must only use function that is supported by all levels of WebSphere Application Server in the cluster. For example, applications that use the commonj.timer.TimerManager resource, which is new in WebSphere Application Server V6, should not be deployed to a cluster including both V5.1 and V6 servers.

18.6.3 Configuring ports

In a coexistence scenario, the ports need to be configured properly to avoid conflicts. It is best to review Port number settings in WebSphere Application Server versions to find port conflicts that might occur when you intend for an earlier version to coexist with WebSphere Application Server V6.

Review the WebSphere Application Server V6 documentation for each version for details about port number settings. In WebSphere Application Server V6 you can change port settings on the port assignment panel while using the Profile creation wizard or use scripting to change the values.

18.6.4 Post-migration configuration

Installation automatically configures WebSphere Application Server V6 and all other bundled products. There is no need for additional configuration if you did not migrate from an earlier version. If you migrate an WebSphere Application Server V4.0.x installation, there are some items to review before considering your environment fully configured.

Examine any Lightweight Third Party Authentication (LTPA) security settings you might have used, and apply the settings in the WebSphere Application Server security settings. Global security that uses LTPA authentication in V4.0.x is migrated to WebSphere Application Server. However, although global security was enabled in V4.0.x, it is disabled during migration to WebSphere Application Server V6.

Check the WASPostUpgrade.log file in the logs directory, for details about JSP 0.91 objects that the migration tools do not migrate. WebSphere Application Server V6 does not support JSP 0.91 objects. The migration tools do not migrate JSP objects configured to run as JSP 0.91 objects. The migration tools do, however, recognize the objects in the output and log them. WebSphere Application Server V6 runs JSP 1.0 and 1.1 objects as JSP 1.2 objects, which is its only supported level.

Identify and use the migration tools to migrate non-migrated nodes in V4.0.x repositories that have multiple nodes. A WebSphere Application Server V4.0.x repository can contain more than one node name and its associated children. The WASPostUpgrade tool processes only those objects and children that match the migrating node. This determination is made by checking the names of nodes in configuration files with fully qualified and non-qualified network names of the migrating machine. Update J2EE resources in client JAR files to the new resource format with the ClientUpgrade tool. J2EE applications might exist on the client, if the client has client JAR files with J2EE resources.

To migrate a WebSphere Application Server V4.0.x or V5.x XML application to the WebSphere Application Server V6, you must recompile it. Configure WebSphere Application Server to use a database. For example, you can configure WebSphere Application Server to use DB2. Review your Java virtual machine settings to verify that you are using a heap size of at least 50 for improved startup performance. If you have used a smaller heap size in the past, you can use the default heap size, which is now 50.

Now you are finished with pre-test configuration. You might have to fine-tune your WebSphere Application Server environment as you test it. Test all redeployed applications before moving them into production.

18.7 Migration troubleshooting

To resolve issues that you encountered in migrating from an older version of WebSphere Application Server, determine whether the issues occur using the pre-upgrade tool or the post-upgrade tool.

There are a number of common issues that can be easily identified:

- When migrating a Deployment Manager, the WebSphere Application Server V6 cell name must match the V5.x cell name. If you create a profile with a new cell name, the migration will fail.
- When migrating a federated node, the new cell name and node name must match their respective WebSphere Application Server V5.x names.
- If you create a profile that does not meet the migration requirements (such as naming requirements), you can remove the old profile and create new one, rather than uninstalling and re-installing the WebSphere Application Server V6 product.
- If you migrate a node to WebSphere Application Server V6, then discover that you need to roll back to V5.x you can use the wsadmin command with specific parameters to roll back from WebSphere Application Server V6.
- Look for these log files and browse them for clues:
  - install_dir/profile/profile_name/logs/WASPostUpgrade.time stamp.log
  - install_dir/profile/profile_name/logs/WASPostUpgrade.trace
  - migration_backup_dir/WASPreUpgrade.time stamp.log
  - install_dir/logs/clientupgrade.time stamp.log

- Look for the following in the migration_backup_dir/WASPreUpgrade.time stamp.log, migration_backup_dir/WASPreUpgrade.time stamp.log, or install_dir/logs/clientupgrade.time stamp.log files:
  - MIGR0259I: The migration has successfully completed.
  - MIGR0271W: The migration completed with warnings.

  If the following message appears, attempt to correct any problems based on the error messages that appear in the log file.

  MIGR0286E: The migration failed to complete.

  After correcting any errors, rerun the command from the bin directory of the product installation root. If the errors persist, rerun the command with trace enabled.
- Look at your trace output to see more detailed message information.

18.8 Summary

The migration to WebSphere Application Server V6 can be accomplished easily with careful planning and forethought. It is important to understand how the new version impacts the current environment and develop a coherent strategy. In many cases, more than one version of WebSphere is in operation and careful planning is needed to avoid issues. New migration tools are available to simplify the migration process but a thorough understanding of the product and new technologies is still necessary. A comprehensive migration strategy and planning created in advance greatly increases the chance of avoiding any migration issues.

Part 3

Appendix

This part provides a sample topology walkthrough that is based on the scenarios that are described in 9.11, “Topology with redundancy of several components” on page 281. It provides a guideline to build a complex topology.

A

Sample topology walk-through

This appendix provides an mapping between one of the topologies illustrated in Chapter 9, “Topologies” on page 247 and the installation and configuration of a working example. It contains the following sections:

- Topology review
- Components installation
- Building the topology
- Testing
- Deployment testing

Topology review

Chapter 9, “Topologies” on page 247 reviewed the available topology options and implemented a variant of one in 9.11, “Topology with redundancy of several components” on page 281. For this wall-through, we did not install a redundant load balancer. There is no technical reason why this option was not selected. Instead, the motivating factor was equipment availability.

Figure A-1 depicts the selected topology. This topology is the one most commonly implemented by IBM clients as discovered through discussions with IBM clients, as well as IBM Global Services teams who are responsible for implementation of WebSphere environments.

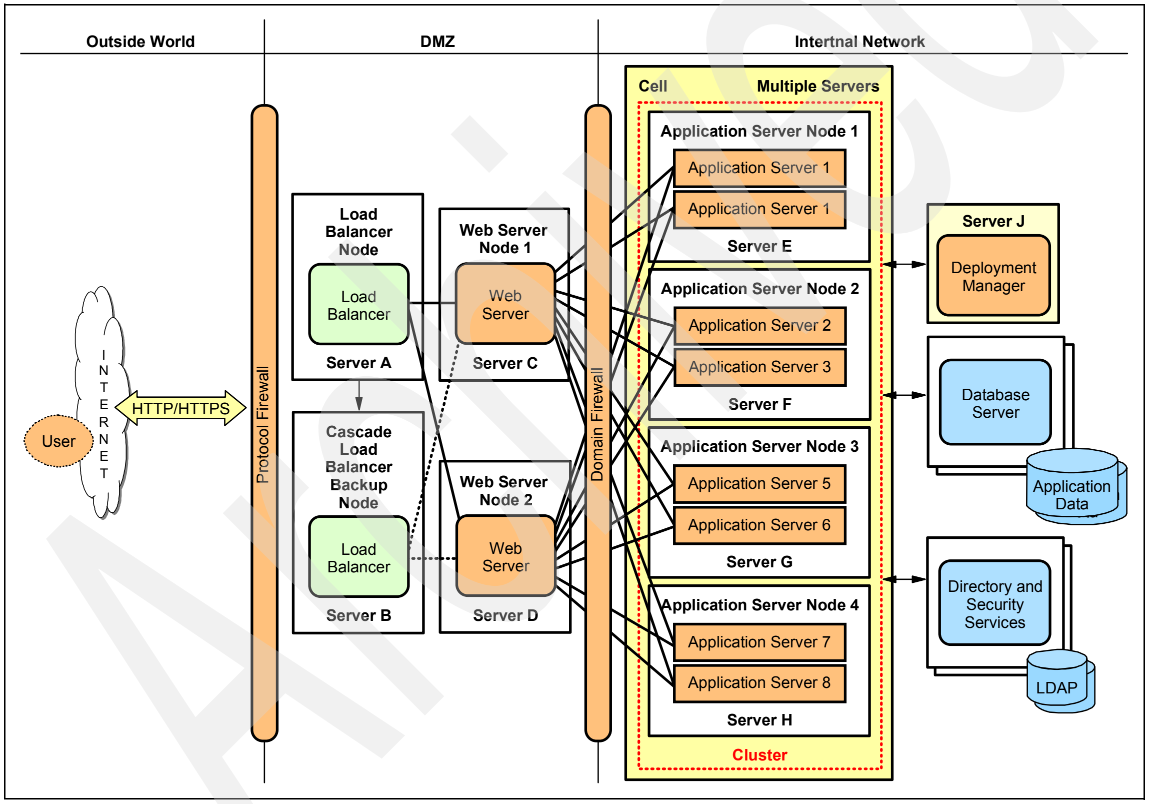


Figure A-1 Topology with redundancy of several components

By reviewing this figure, you can see that this topology isolates each tier. Web servers are separated from application servers, data servers and security servers are isolated from one another, and the Deployment Manager is a separate machine. This configuration provides the greatest resiliency of a site, and the greatest administrative complexity. From that topology, the following summarize the advantages and disadvantages of this implementation:

Advantages

This topology is designed to maximize performance, throughput and availability. It incorporates the benefits of the other topologies already discussed.

- Single point of failure is eliminated from the Web server, application server, database server, and LDAP server, due to redundancy of those components.
- It provides both hardware and software failure isolation. Hardware and software upgrades can be easily handled during off-peak hours.
- Horizontal scaling is done by using both the IP sprayer (for the Web server nodes) and the application server cluster to maximize availability.
- Application performance is improved by using several techniques:
  - Hosting application servers on multiple physical machines to boost the available processing power.
  - Using clusters to vertically scale application servers which makes more efficient use of the resources of each machine.
- Applications with this topology can make use of workload management techniques. In this example, workload management is performed through:
  - WebSphere Edge Component Network Dispatcher to distribute client HTTP requests to each Web server.
  - WebSphere Application Server Network Deployment workload management feature to distribute work among clustered application servers.

Disadvantages

This combined topology has the following disadvantages:

- Isolating multiple components and providing hardware redundancy increases the overall cost of this implementation.
- Licenses must be obtained for all the machines to be used, and this can add significant costs to provide resiliency.
- This is an example of a standard implementation, but it should also be noted that it is one of the most complex. The costs of administration and configuration work should be pondered in relation to benefits of increased performance, higher throughput and greater reliability.

Figure A-2 on page 534 shows the completed topology.

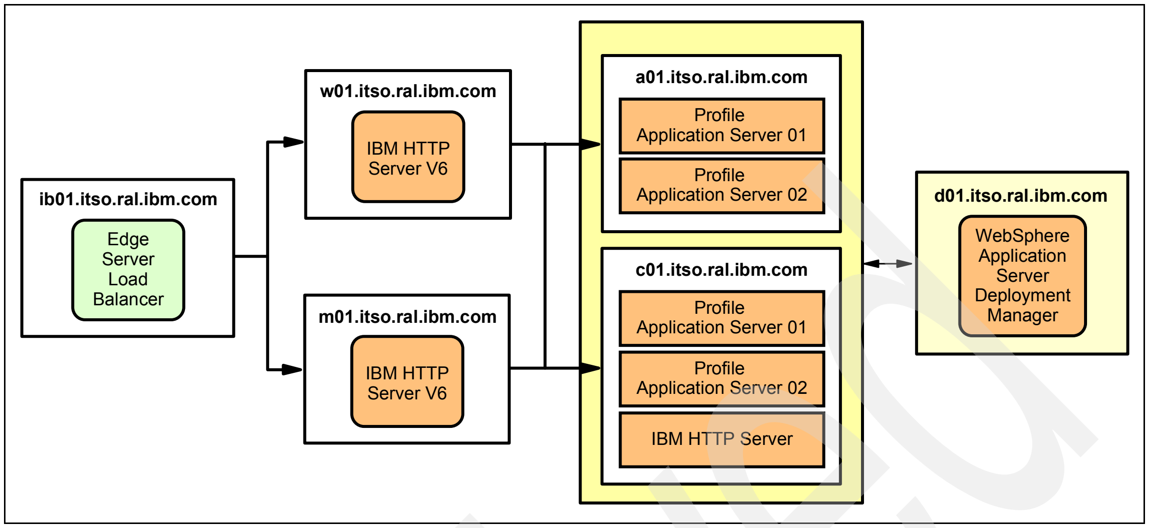
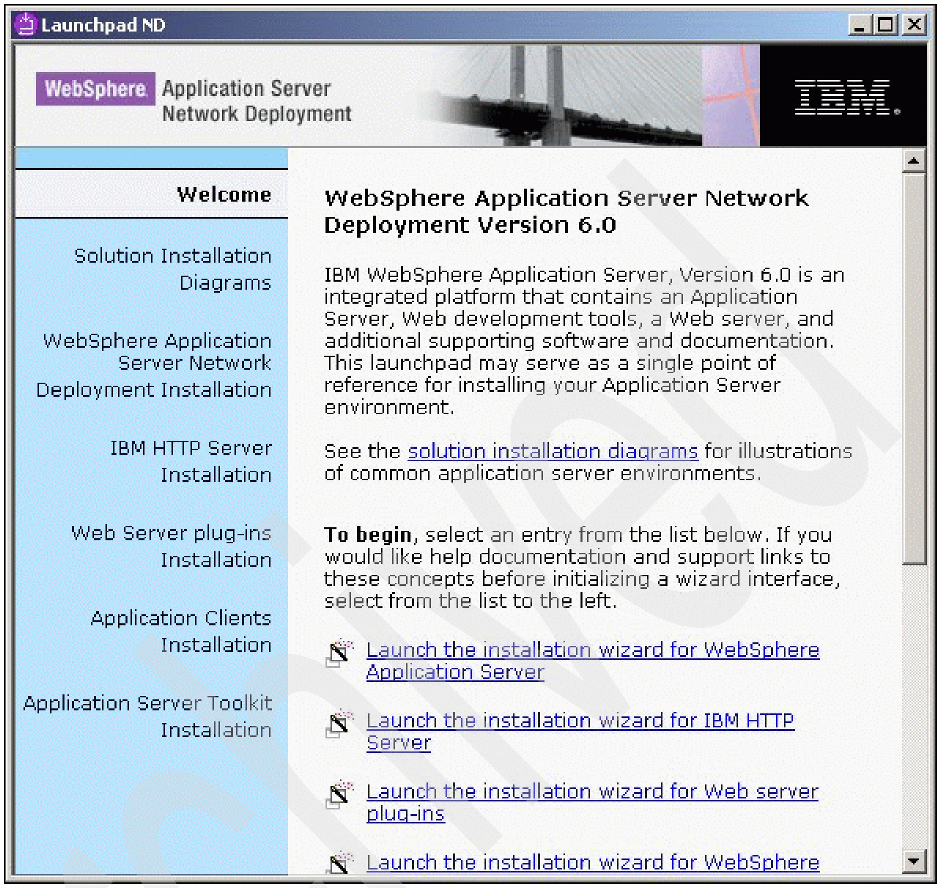

ib01.itso.ral.ibm.com
Edge Server Load Balancer
w01.itso.ral.ibm.com
IBM HTTP Server V6
m01.itso.ral.ibm.com
IBM HTTP Server V6
a01.itso.ral.ibm.com
Profile Application Server 01
Profile Application Server 02
c01.itso.ral.ibm.com
Profile Application Server 01
Profile Application Server 02
IBM HTTP Server
d01.itso.ral.ibm.com
WebSphere Application Server Deployment Manager

Figure A-2 Completed Topology

Components installation

This example topology involves isolating as many of the components as possible. For the purposes of this example, we did not, however, choose to implement all of the components in the topology. Our decisions were based solely on availability of equipment and resources, not a technical reason. When completed, our topology should resemble Figure A-2.

The remainder of this appendix works through the process of building out the selected topology. The following sections detail the process used to install and configure each item. The installation is started through the launch pad, which is shown in Figure A-3 on page 535.


Launchpad ND
WebSphere Application Server Network Deployment
IBM.
Welcome
Solution Installation Diagrams
WebSphere Application Server Network Deployment Installation
IBM HTTP Server Installation
Web Server plug-ins Installation
Application Clients Installation
Application Server Toolkit Installation

WebSphere Application Server Network Deployment Version 6.0

IBM WebSphere Application Server, Version 6.0 is an integrated platform that contains an Application Server, Web development tools, a Web server, and additional supporting software and documentation. This launchpad may serve as a single point of reference for installing your Application Server environment.

See the solution installation diagrams for illustrations of common application server environments.

To begin, select an entry from the list below. If you would like help documentation and support links to these concepts before initializing a wizard interface, select from the list to the left.

Launch the installation wizard for WebSphere Application Server

Launch the installation wizard for IBM HTTP Server

Launch the installation wizard for Web server plug-ins

Launch the installation wizard for WebSphere

Figure A-3 WebSphere Application Server V6 installation Launchpad

The installation process for all the components is fairly similar and all the installations are started from a single point, the Launchpad. Because this example uses WebSphere Application Server Network Deployment V6, at the end of the installation we launched manually the profile creation wizard.

Application server nodes

This configuration is one in which WebSphere Application Server V6 is installed on a node in a stand-alone configuration. This implies that the machine and the application servers defined on this machine would not be used (initially) in a cell.

From the Launchpad, we selected **Launch the Installation wizard for WebSphere Application Server** and followed the prompts. We used this to install two nodes from Figure A-2 on page 534. The nodes c01.itso.ral.ibm.com and a01.itso.ral.ibm.com were built as single-tier installation servers. As part of the installation, a step new to WebSphere Application Server V6 is the profile creation. We created an application server profile for both nodes. This profile installed server1 and the sample applications.

The purpose of doing this installation was mainly for testing, in the event of finding any problem, the simpler the installation, the easier to do problem determination.

Deployment Manager node

This is the Deployment Manager configuration installation. From the Launchpad, we chose **WebSphere Application Server Network Deployment Installation** from the left menu options and then selected **Launch the installation wizard** for the WebSphere Application Server Network Deployment from the main panel. This is another way to start the Network Deployment installation from the Launchpad. This completed the install of the node d01.itso.ral.ibm.com, shown in Figure A-2 on page 534. Using the profile wizard, WebSphere Application Server V6 is configured with a Deployment Manager profile, allowing this node to administer a cell and the resources associated with a cell. In our environment, it was used to create the cell used to manage the Web servers, and the application servers we have defined.

Tip: Nodes that are federated need to have their system clocks very closely in sync. This does not mean in the same time zone, but that they must be within five minutes of each other with respect to time zone variance. If they are not, the node will not federate. If the node is federated and then the gap grows to greater than five minutes, the node will not be able to synchronize correctly with the Deployment Manager repository.

IBM HTTP Server V6

This is the new version of the IBM HTTP Server. It is based on the GNU Apache build 2.0.47. We selected three nodes to install on. The first is c01.itso.ral.ibm.com. This node is a WebSphere Application Server node, but we wanted to be able to verify the plug-in locally, so we installed an IBM HTTP Server V6 server on the machine as well. The second and third node are m01.itso.ral.ibm.com and w01.itso.ral.ibm.com. These two nodes are to be used as HTTP Server nodes exclusively, and communicate to the cell through the plug-in module only. We have also added the nodes to the cell as unmanaged nodes, which allows us to propagate changes to the plugin-cfg.xml through the console, a feature new to WebSphere Application Server V6.

WebSphere Edge Server Load Balancer

Version 6 is the updated version for the WebSphere Edge Server components, which includes the Load Balancer and Caching Proxy components. We chose to use the Load Balancer component so that we could distribute traffic between the three Web servers, m01.itso.ral.ibm.com, c01.itso.ral.ibm.com and w01.itso.ral.ibm.com. Caching Proxy would typically be used for segmenting traffic, as indicated in “Segmenting the workload” on page 204, but was not implemented for this testing. A single cluster was created that supported port 80 load balancing. We chose not to implement a port 443 cluster, because we did not had the time to configure the SSL keystores and certificates. Referring to Figure A-2 on page 534, the machine that was our load balancing machine was named lb.itso.ral.ibm.com.

Building the topology

This section covers the process that was used to logically build out the topology. The purpose is to provide a sample method for verifying the components while building the topology. We point out items that are new to WebSphere Application Server V6 implementations so that you can get a feel for what has changed between versions.

WebSphere profiles

A new feature to WebSphere Application Server V6 is the profile. Now that the code installs to the same path, the profile was used to define how a node would be used when installed. The profile creation wizard offers three types of profiles:

- Deployment Manager profile, which used to manage a Network Deployment cell.
- Application Server profile, which creates a stand-alone machine. This can then be used to run applications on a single machine, or be federated into a cell at a later time. This profile adds all of the sample applications, data sources, system applications, and so forth.
- Custom profile, which is an empty node. No servers, samples, or administrative applications are available through this profile. The purpose would be to immediately federate the node into a Network Deployment cell.

If a node is installed, but a profile has not been created, it simply does not work. Our example lists which profile option we chose when performing that portion of the topology building.

Building the application server nodes

To begin, we installed the first application server node. Because this was a test environment, we did not deviate from the standard installation options and did use the GUI for the installation. Refer to *WebSphere Application Server V6 - System Management and Configuration Handbook*, SG24-6451 for details about silent installations and response file usage.

After the installation was completed, we then launched the profile creation wizard to configure the profile. We chose the Application Server profile as listed above, so that we would be able to test the individual components first. We also, as part of this build, installed the HTTP Server onto the node locally to allow us to perform one additional test as part of our node verification. Because we wanted to assert the communication between the Application Server and the HTTP Server, we also installed the binary plug-in modules so that the Web server would hand off requests to the Application Server.

To test the node, we performed two basic tests. First, we ran the Installation Verification Test (IVT) from the First Steps console. When this returned a valid state for the machine, we were ready to perform the second test. This test was to start server1 and the sample applications. After the server started, we ran tests using the embedded HTTP server on the node to verify that the applications functioned correctly. Our tests included snoop, and Plants By WebSphere sample applications.

When the basic tests were completed, we wanted to verify the complete node functionality. For this test, we used the Web server we installed on the application server node, c01.itso.ral.ibm.com. We generated a copy of the plugin-cfg.xml and made sure it was in the correct location for the HTTP Server, based on the default configuration of the module, and then restarted the Web server. At that point, we ran the same tests against snoop and Plants By WebSphere to verify that the Application Server was receiving requests correctly through the plug-in. When we were satisfied, it was time to begin the logical work for the remaining components.

Building a remote HTTP server

Now that we had verified a working configuration of an application server node, as well as a stacked HTTP server on the same node, it was time to install our first remote HTTP server and test. We used the system defaults for this install. When the HTTP server was installed, we then followed the prompts to install the binary plug-in module. This step allowed us to add the plug-in, as well as generate the script required to add the node to a cell for remote plug-in management.

After the installation was completed, we ran two functional tests to verify the operation of the node. First, we started the HTTP server, and then verified using an external browser we were able to load the default splash page. Then, we started the administration server and verified it was listening correctly on the administration port. Because the administration engine requires the user to log in, we were also required to create an entry in the access file for the administration server. This is done using the **htpasswd** command in the HTTP server bin directory. After we had a user name and password to sign in, we were then able to load the administration URL in a browser, and verify it prompted for a user and password. We then attempted to use the ID we created, and verify that we had inserted the record correctly into the access file. This completed functional tests, and it was now time to verify the plug-in configuration worked as well.

To verify the plug-in installation, we copied the plugin-cfg.xml from the application server node, c01.itso.ral.ibm.com, to the local node to the location stated in the httpd.conf. We restarted the Web server, and then tested loading the snoop servlet and Plants by WebSphere using the remote Web server instead of the embedded HTTP server for WebSphere, or the stacked HTTP server on the Application Server. When these tests were successful, we felt comfortable to proceed to the next step.

Building the Deployment Manager

Now that we have installed, configured, and tested an application server node and an HTTP server node, it was time to begin the process of creating the cell environment. Because our node, d01.itso.ral.ibm.com, had been configured to be a Deployment Manager profile, we should be able to start federating the Application Server and adding the Web servers to the console for remote management. First, we need to verify the installation is valid. To do so, we start with the installation verification test through the First Steps console. When this returns a valid installation, we then attempt to start the Deployment Manager process and launch the console. After we are able to load the console and log in, we can assert that the Deployment Manager function is tested and working correctly.

Adding Web servers to the administrative console is a new feature in WebSphere Application Server V6, which allows through the administration port the ability to propagate plugin-cfg.xml changes to the Web servers. This offers a significant improvement over the old method, which required copying the plugin-cfg.xml to each Web server.

To verify our configuration, we first made sure that server1 was running on our application server node, c01.itso.ral.ibm.com. We then launched the administrative console for the Deployment Manager, and attempted to add the node through the System Administration menu path. Node federation creates log entries on the node to be federated, so any issues with the federation would be logged to the Application Server node. When the federation was verified to complete, we then removed the node from the console and saved the configuration again. We verified that the cell was again empty, and then ran the addNode script from the Application Server node to verify that functionality as well. After that was completed, we were able to move on to adding a Web server node to the cell.

Adding a Web server node is done in one of two ways. When installing the plug-in modules, a script is created on the HTTP server node that performs a **wsadmin** task to create the appropriate node entries. You can simply copy this file to the Deployment Manager node and execute the script. Alternatively, you can use the administrative console to add the node there. Either method requires that you go in afterward and add the authentication information, which is the user ID and password we configured for the administration interface in , “Building a remote HTTP server” on page 538.

Multiple profiles on the same physical machine

With the successful testing of the first federated profile, we were now ready to begin the remaining work on the first application server node, c01.itso.ral.ibm.com from Figure A-2 on page 534. As you can see in the diagram, the intent is to create two distinct profiles on this machine, and have both federated into the Deployment Manager cell. To start, we began with creation of the second profile. Launching the profile wizard, we created a second profile using the prompted defaults. Because this was a second profile, the wizard knew to make sure there were no port conflicts for the second profile, and displayed a dialog box allowing us to verify the ports, and make changes as needed.

When the second profile had been created successfully, we were then ready to begin testing and integration into the cell. First, we used the IVT tool to assert that the second profile functioned correctly. When that test passed, we then verified the administrative console worked for the second profile.

Note: The administrative port for the second instance was changed from the base 9060 to 9061, because the profile wizard detected the first profile, but did not know whether or not the first profile had been federated. If we had used port 9060, we would have received a 404 error in the browser.

With that testing completed, we were ready to verify the node could be used to serve applications through a plug-in correctly. We generated the plug-in for the second profile, and then copied this profile to the Web server m01.itso.ral.ibm.com. We then were able to verify that the snoop and Plants by WebSphere samples did function correctly against the new profile.

When we had established that the profile was a valid installation, it was time to federate the second profile into the cell. For this profile, we chose to use the console only, because the first profile was tested using the administrative console and the addNode command, we did not feel it was necessary to test the federation of the second profile using both methods.

Note: Just as we had to change the administrative port, so did we have to change the SOAP port. The default port 8880 was used by the first profile. To federate the second profile, we needed to change the port in the console to connect to port 8881.

Clustering federated application server nodes

Now that we have successfully federated the second profile, we want to test a cluster built against the two profiles. To do so, we created a cluster named test, and added two servers to the cluster; AppSrv01 and AppSrv02. We then changed the module mapping for the Default Application to use the cluster instead of the original stand-alone server.

After we had the module mapping set correctly, we were then able to generate the plugin-cfg.xml from the command line, and copy it to the remote server m01.itso.ral.ibm.com. We then tested our applications, snoop, and Plants by WebSphere to verify the applications still served correctly.

Integrating HTTP servers

Now that we have verified a functioning cluster, and that it serves through a remote HTTP Server, we should now begin to add the HTTP server to the cell. This is a feature new to WebSphere Application Server V6, that allows the administrator to manage and propagate the plugin-cfg.xml for the Web servers used by this cell. To start, you must add an unmanaged node to the cell. This is done through the System Administration → Nodes section of the console. It takes two arguments, the label for the node (the name that will be seen in the console for the node) and the IP address or host name for the node.

When you have defined the unmanaged node, you then need to go to **Servers** → **Web servers** in the administrative console. From there, select new to add the new Web server. There are four steps for completing the Web server definition:

1. Assign a name and select the node from the list of nodes running the Web server.

2. Enter the properties for the Web server. This includes the type of Web server, the path to the Web server executables, the service name (if running on an Windows machine) and the full path to the plugin-cfg.xml. This step also defines the administration port, as well as the user name and password for the administration access if you are using IBM HTTP Server.
3. Select the Web server template. There is a single template that is used to define the supported HTTP Server. Make sure you select the correct template for your Web server.
4. Verify and create the Web server definition for the console.

Note: WebSphere Application Server V6 offers a new functionality, managing the plugin-cfg.xml via the administrative console. This management now includes propagating the file to the Web server nodes managed in the cell. In order for this configuration to work correctly, you must make sure that you map the Web modules to the Web servers in the console you expect to serve the application.

At this point, there is no need to regenerate the plug-in. As you deploy new applications, however, there is a two step process. First, you must generate the plug-in for the Web server in question. The second step is to then propagate the new plugin-cfg.xml to the Web server. Plug-in propagation causes the Web servers to reload the file, unless the automatic reload is disabled.

Now that we have defined a Web server in the console for m01.itso.ral.ibm.com from Figure A-2 on page 534, we need to make sure that the Web server was also a target for any Web modules installed, so that the plug-in generation from the administrative console works correctly.

Adding the redundant resources

After we had asserted our installation for a single Web server, a single application server node with two profiles, and the Deployment Manager, we incorporated the remaining servers. Our topology calls for two application server nodes running two profiles and two remote Web servers. Using the same processes from the first set of servers, we added a01.itso.ral.ibm.com and w01.itso.ral.ibm.com to the cell. This gave us the final infrastructure that we needed for the topology. This provided a fairly standard configuration for serving applications. The Web servers could easily be placed in a demilitarized zone (DMZ) environment, and the application servers placed behind a firewall. All that remains is configuration of the Load Balancer to spray traffic to the two Web servers.

Load balancer configuration

Configuring the Load Balancer can be done in one of two ways; using the graphical console, or using the command line. For this test, we chose to use the graphical console. To invoke the console, execute **`lbadmin`** from a prompt, or click the icon which runs this same script. If you prefer the command line, you can issue **`dscontrol`** from a prompt, and enter the commands there.

The first step was to create a cluster. This cluster is the host name and IP combination that are used by clients to access the Web site and guarantee the load balancing. When creating the cluster, the console prompts you for the interface to which to bind this cluster. Select the primary interface as defined for that machine. It is customary for the name of the cluster (which is simply a label) to be the DNS name for the cluster, for ease of administration. After the cluster has been created, you add a port to the cluster. For our purposes, we chose HTTP, which defaults to port 80. The next step is to add the servers that will be serving for this cluster. We simply added m01.itso.ral.ibm.com, w01.itso.ral.ibm.com, and c01.itso.ral.ibm.com. We used this name for both the label and the host name, so that we could verify that the Load Balancer could resolve the names correctly to an IP.

Note: The dialog box in the console has an option for network router address and a check box to indicate it must be used. This should only be the case if your Web server is not on the same network as the Load Balancer. For the purposes of our testing, all the machines were on the same logical network.

Now that the Web servers are defined to the cluster, we must go to the Web servers and verify the configuration to assure they can serve that cluster address. When using IBM WebSphere Edge Server Load Balancer, you are required to bind the cluster IP as an alias to the loopback adapter for each Web server, and the load balancer itself. Refer to specifics for your operating system for adding a loopback alias.

Important: Windows operating systems require a special network adapter be installed to handle loopback addressing. Also, Windows adds a bad route for the loopback alias, which then must also be removed prior to verifying that the server is functioning correctly. Refer to *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392 for details about the adapter and the bogus route.

At this point, we are now ready to enable the advisor. The Load Balancer uses a port advisor to verify the health of servers participating in the cluster. In the case of HTTP requests, the advisor sends a HEAD request to each of the servers participating in the cluster. If a server does not respond correctly to the advisor, the advisor marks the node down in the Load Balancer cluster so that traffic is not sprayed to the malfunctioning server.

To do unit testing of the Load Balancer cluster, we simply marked down two of the three servers in sequence, and then verified that the cluster address was still serving from the remaining Web server. We repeated this test until it was asserted that each of the three Web servers correctly served the Load Balancer cluster.

Testing

Now that the entire environment has been built and unit tested, we now want to verify expected functionality. To do this, we created a simple frameset HTML page that was placed onto each one of the Web servers. The top portion of the frame indicated which Web server was serving the request for the cluster IP, the lower frame executed the snoop servlet, which shows which application server responded to the plug-in request. If you look at Table A-1, you can see the various page loads, and the results of those loads. The table lists in order the page request, and which Web server and application server responded to the two frames. The tests were continued until it was verified that each Web server processed a request from each of the four application server instances.

Table A-1 Testing the entire environment

Web Server Host	**Application Server Host**	**Server Profile**
w01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv01
w01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv01
c01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv01
m01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv02
w01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv02
c01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv02
m01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv01
c01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv01
w01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv02
c01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv02
w01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv02
m01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv01

Web Server Host	Application Server Host	Server Profile
w01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv01
m01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv02
w01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv02
c01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv02
m01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv01
c01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv02
m01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv01
w01.itso.ral.ibm.com	c01.itso.ral.ibm.com	AppSrv01
c01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv01
m01.itso.ral.ibm.com	a01.itso.ral.ibm.com	AppSrv02

One item of interest is that the operation of the Load Balancer is such that it uses a statistical round robin when the advisor is running, which means that the Web servers responded based on the metrics the Load Balancer used to weight the likelihood a server would be able to respond to the request. Even after turning off the HTTP port advisor and the manager, the weights assigned to the servers based on response times continued to keep the clustered Web servers from serving in a true round robin fashion.

After testing at this level, we were confident that the environment was behaving as expected. Each Web server received some of the traffic from the Load Balancer, and each application server served some of the requests to the snoop servlet. It was now time to begin using the environment to verify new functionality in the WebSphere Application Server Network Deployment V6 environment.

Deployment testing

So far we have built a WebSphere Application Server V6 environment with multiple application servers which are part of a administrative cell and which are clustered. We now look at how to go about deploying applications to this topology.

The deployment objective

This section walks through the installation of an enterprise application called RedbookApplication. The objective is to deploy the RedbookApplication to all servers on the cluster that we have created. The application will be then served as one application across all these servers.

The choice of deployment method

The first decision to be made is which deployment method should be used. Remember that there are five options for deploying applications to a WebSphere Application Server V6 environment:

- WebSphere administrative console
- **`wsadmin`**
- **`Ant`**
- WebSphere Rapid Deployment
- Application Server Toolkit

WebSphere Rapid Deployment

The WebSphere Rapid Deployment option is not really for production environments. It is for deploying applications to application servers and then updating the applications by dropping the install package into a monitored directory. If the application package is removed from this monitored directory then the application is uninstalled from the application server, this is not something that you would want to occur in a production environment.

More over, WebSphere Rapid Deployment is only for deploying applications to specific servers. It works for deploying to stand alone servers and also to federated servers (application servers that are part of a cell). But WebSphere Rapid Deployment cannot deploy applications to application servers that are members of a cluster. As all of our application servers are cluster members, we did not therefore use WebSphere Rapid Deployment as our deployment method for this topology.

Application Server Toolkit

The Application Server Toolkit is generally a tool for assembling and deploying tools for testing and profiling purposes, again this tool cannot be used to deploy to a clustered server. Therefore, we did not select this tool for deploying our application.

The wsadmin and Ant tools

We could very well use **`wsadmin`** commands or scripts to deploy the application. We could also use **`Ant`** script calling WebSphere task blocks or ANT invocations of wsadmin to deploy our application. However, we do not necessarily need to run these tasks often and therefore writing a script might not be worth our time.

WebSphere administrative console

We deployed the application using the WebSphere administrative console of the Deployment Manager. It should be noted that the general steps that we followed using the WebSphere administrative console would be the steps that you would need to follow when using `Ant` or `wsadmin` commands and scripts too. We installed the RedbookApplication which is contained in an Enterprise Archive file called RedbookApplication.ear.

Steps for deployment

The following are the steps for installing the RedbookApplication via the WebSphere administrative console to a cluster of application servers (called *RedCluster* in this example):

1. Load the WebSphere administrative console into the browser.
2. Log in.
3. Select **Applications** from the left-hand menu.
4. Select **Install Application** from the left-hand menu.
5. Select **Local File System** or **Remote File System**, depending on where the EAR file is.
6. Click **Next** to upload the EAR file.
7. On the page entitled *Preparing for the application installation*. accept all defaults because you do not want bindings overridden and you do want deployment to the default host.
8. On the page entitled *Install New Application - Select installation options*, accept all defaults for this application.
9. On the page entitled *Install New Application - Map modules to servers*, define where the EJB module and Web module components are going to run. In this case, you want all modules to run on all of the application servers in the cluster. You also want each of the three Web servers to be configured to serve the application. Therefore, select the cluster (called RedCluster in this example) and the Web servers. Check the check boxes next to the EJB and WAR projects and then click **Apply**. The modules and servers are now mapped as you want them to be. Click **Next**.
10. On the page entitled *Install New Application - Provide JNDI Names for Beans***,** accept all defaults for this application's EJB JNDI mappings.
11. On the page entitled *Install New Application - Map virtual hosts for Web modules*, check the box to deploy the Web module against the default host, and click **Next**.

12. On the page entitled *Install New Application - Ensure all unprotected 2.x methods have the correct level of protection*, accept all defaults because you do not want to protect your EJB methods. Click **Next**.
13. Inspect the Summary page and click **Finish**.
14. A summary of the installation appears and informs you of the steps being executed and of the installations success.
15. At this point, click **Save the configuration**. Select **Synchronize changes with nodes** and click **Save**.
16. Go to Clusters in the left-hand menu and ensure that RedCluster is started.
17. Go to Application Servers in the left-hand menu and ensure that all cluster member application servers in the RedCluster have started. You might have to wait while these servers are started in the cluster.
18. Go to Servers and select all the Web servers you find there. Click **Generate Plug-in**. Select the Web servers again and click **Propagate Plug-in**. The Web servers should now all know the relevant details of the Redbook Web application.
19. Check that you can access the application via the address for the load balancer cluster and the context root and servlet path for the application. We load this in our browser and can see and use the Redbook Web application.

Abbreviations and acronyms

ACL	Access Control List
API	Application Program Interface
ASP	Active Server Pages
AST	Application Server Toolkit
BCI	Byte-code Instrumentation
BPEL	Business Process Execution Language
CA	Certification Authority
CCI	Common Client Interface
CICS	Customer Information Control System
CMP	container-managed persistence
CMS	Code Management System
CORBA	Common Object Request Broker Architecture
CPD	Client Performance Data
CSIv2	Common Security Interoperability Version 2
CVS	Concurrent Versions System
DASD	direct-access storage device
DCS	Distribution and Consistency Services
DII	Dynamic invocation interface
DMZ	Demilitarized Zone
DOM	Document Object Model
DRS	Data Replication Service
DTD	Document Type Definition
EAR	Enterprise archive
EDI	Electronic Data Interchange
EGL	Enterprise Generation Language
EIS	Enterprise Information Systems
EJB	Enterprise JavaBean
FDL	Flow Definition Language
FDML	Flow Definition Markup Language
FRCA	Fast Response Cache Accelerator
GUI	Graphical User Interface
HACMP	High Availability Cluster Multi-Processing (AIX)
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IHS	IBM HTTP Server
IIOP	Internet Inter-Orb Protocol
IMS	Information Management System
ISV	Independent Software Vendor
ITSO	International Technical Support Organization
J2EE	Java 2 Platform Enterprise Edition
J2SE	Java 2 Platform, Standard Edition
JAAS	Java Authentication and Authorization Service
JACC	Java Authorization Contract for Containers
Jacl	Java Command Language
JAF	JavaBean Activation Framework
JAXP	Java API for XML Processing
JAXR	Java API for XML Registries

JAX-RPC	Java API for XML-Based RPC
JCA	J2EE Connector Architecture
JCP	Java Community Process
JDBC	Java Database Connectivity
JIDL	Java Interface Definition Language
JMS	Java Message Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRE	Java Runtime Environment
JSF	JavaServer Faces
JSP	JavaServer Pages
JSR	Java Specification Request
JSSE	Java Secure Sockets Extension
JSTL	JavaServer Pages Standard Tag Library
JTA	Java Transaction API
JVM	Java Virtual Machine
LPAR	Logical Partition
LTPA	Lightweight Third Party Authentication Protocol
MDB	Message-driven beans
MIME	Multipurpose Internet Mail Extensions
MTBF	mean time between failure
MTTR	mean time to recovery
MVC	Model, View, Controller pattern
NAS	Network Attached Storage
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OO	Object Oriented
ORB	Object Request Broker
PAM	Pluggable Authentication Module
PCT	Profile Creator Tool
PKI	Public Key Infrastructure
PME	Programming Model Extensions
PMI	Performance Monitoring Infrastructure
QCF	queue connection factory
QoS	Quality of Service
RACF	Resource Access Control Facility
RAD	Rational Application Developer
RAID	Redundant Array of Independent Disks
RAR	Resource Adapter Archive
RAS	Reliability, Availability, and Serviceability
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RWD	Rational Web Developer
SAAJ	SOAP with Attachments API of Java
SAM	Software Asset Management
SAN	Storage Area Network
SAS	Secure Authentication Service
SAX	Simple API for XML
SCCS	Source Code Control System
SCM	Source Code Management
SCSI	Small Computer System Interface
SDO	Service Data Object
SEI	service endpoint interface
SFSB	Stateful Session EJBs
SLA	Service Level Agreements

SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
SPI	Service Provider Interface
SPOF	Single Point of Failure
SSL	Secure Socket Layer
SSO	single signon
SWAM	Simple WebSphere Authentication Mechanism
TAI	Trust Association Interceptor
TAM	Tivoli Authorization Module
TCF	Topic Connection Factory
Tcl	Tool Command Language
TPV	Tivoli Performance Viewer
UDR	UDDI Business Registry
UCA	Unicode Collation Algorithm
UCM	Unified Change Management
UDDI	Universal Description, Discovery and Integration
UDDI4J	Java-based Open Source UDDI V2 client/library
UML	Unified Modeling Language
URI	Uniform Resource Locators
URL	Universal Resource Locator
UUID	Universal Unique Identifier
UUT	UDDI Utility Tools
VOB	Versioned Object Bases
WAR	Web Application Archive
WLM	Work Load Manager
WS	Web Services
WSDL	Web Services Description Language
WSEE	Web Services for J2EE
WSIF	Web Services Invocation Framework
WSFL	Web Services Flow Language
WS-I	Web Services Interoperability
WSS	Web Service Security
XML	Extensible Markup Language
XSD	XML Schema Definition

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 555. Note that some of the documents referenced here might be available in softcopy only.

- *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392
- *WebSphere Version 5.1 - Application Development Handbook*, SG24-6993
- *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891

Online resources

These Web sites and URLs are also relevant as further information sources:

- XDoclet, Attribute Oriented Programming.

  `http://xdoclet.sourceforge.net/xdoclet/`
- Java Community Process, Community Development of Java Technology Specifications.

  `http://www.jcp.org`
- J2SE 5.0

  `http://java.sun.com/j2se/1.5.0`
- Tcl Developers Xchange.

  `http://www.tcl.tk/software/java/`
- WebSphere Application Server Information Center.

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`
- WebSphere Application Server - Support

  `http://www.ibm.com/software/webservers/appserv/was/support/`

- The Apache Ant Project

  http://ant.apache.org
- WebSphere Application Server supported hardware and software.

  http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html
- IBM HTTP Server Information Center.

  http://www.ibm.com/software/webservers/httpservers/doc/v2047/manual/ibm/en_US/
- The WSFL specification.

  http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
- The Web services insider: Introducing the Web services Flow Language.

  http://www.ibm.com/developerworks/webservices/library/ws-ref4/index.html
- The Web services insider: Getting into the Flow.

  http://www.ibm.com/developerworks/webservices/library/ws-ref5/
- Business process execution language for Web services, Version 1.0.

  http://www.ibm.com/developerworks/webservices/library/ws-bpel1/index.html
- Business processes in a Web services world.

  http://www.ibm.com/developerworks/webservices/library/ws-bpelwp/
- IBM business process execution language for Web services Java runtime (BPWS4J).

  http://www.alphaworks.ibm.com/tech/bpws4j
- XLANG— Web services for business process design.

  http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
- (BPEL4WS) 1.0 specification.

  http://www.ibm.com/developerworks/library/ws-bpel/
- J2EE Connector Architecture Product Information.

  http://java.sun.com/j2ee/connector/products.html
- The Source for Developers, Java Message Service (JMS).

  http://java.sun.com/jms
- DB2 Information Integration site.

  http://www.ibm.com/software/data/integration
- Information integration, Distributed access and data consolidation white paper.

  ftp://ftp.software.ibm.com/software/data/pubs/papers/etl.pdf

- JMS Application Architectures.

  `http://www.theserverside.com/articles/article.tss?l=JMSArchitecture`
- Enterprise Generation Language - Documentation.

  `http://www.ibm.com/developerworks/websphere/zones/studio/egldocs.html`
- Rational software overview.

  `http://www.ibm.com/software/rational`
- Concurrent Versions System.

  `http://www.cvshome.org`
- Subversion versioning project.

  `http://subversion.tigris.org`
- Java Specification Requests overview.

  `http://www.jcp.org/en/jsr/all`
- The Apache Maven Project.

  `http://maven.apache.org`
- IBM Rational Unified Process product overview.

  `http://www.ibm.com/software/awdtools/rup`
- JUnit testing resources site.

  `http://www.junit.org`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

WebSphere Application Server V6 Planning and Design, WebSphere Handbook Series

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

WebSphere Application Server V6 Planning and Design

WebSphere Handbook Series

Discusses end-to-end planning for WebSphere implementations

Provides best practices

Includes a complex topology walk-through

This IBM Redbook discusses the planning and design of industrial strength WebSphere Application Server V6 environments. The content of this redbook is oriented to IT architects and consultants who require assistance when planning and designing from small to large and complex implementations.

This redbook addresses the new packaging and features incorporated in WebSphere Application Server V6, including technology options, product architecture, and integration with other products. It also covers the most common implementation topologies and addresses in detail the planning for specific tasks and components that conform to the WebSphere Application Server environment.

SG24-6446-00 ISBN 0738492183