WebSphere software

IBM

Redbooks Paper

Carla Sadtler

WebSphere Application Server V6.1: Technical Overview

WebSphere® Application Server is the implementation by IBM® of the Java™ 2 Enterprise Edition (J2EE™) platform. It conforms to the J2EE 1.4 specification. WebSphere Application Server is available in unique packages that are designed to meet a wide range of customer requirements. At the heart of each package is a WebSphere Application Server that provides the runtime environment for enterprise applications.

This discussion centers on the runtime server component of WebSphere Application Server.

WebSphere Application Server packaging

WebSphere Application Server comes in several packaging options. In addition to the application server component, each package contains an appropriate combination of complementary products (for example, IBM HTTP Server, Application Server Toolkit, and Edge components).

Distributed platforms

WebSphere Application Server V6.1 has the following packaging options for distributed platforms, including IBM AIX®, HP-UX, Linux®, Solaris™, and Microsoft® Windows®:

- IBM WebSphere Application Server - Express V6.1, referred to as *Express* (target availability: fourth quarter 2006)
- IBM WebSphere Application Server V6.1, referred to as *Base*
- IBM WebSphere Application Server Network Deployment V6.1, referred to as *Network Deployment*

Packaging information for Base and Network Deployment can be found at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rtop_overview.html

The home page for WebSphere Application Server on distributed platforms can be found at:

http://www-306.ibm.com/software/webservers/appserv/was/index.html

System z

For WebSphere Application Server on System z™, the following edition is available:

- IBM WebSphere Application Server for z/OS® V6.1, a full-function version of the Network Deployment product

Packaging information for WebSphere on System z can be found at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.zseries.doc/info/zseries/ae/rtop_overview.html

The home page for WebSphere Application Server for z/OS can be found at:

http://www-306.ibm.com/software/webservers/appserv/zos_os390/

System i

WebSphere Application Server on System i™ has the following packaging options:

- WebSphere Application Server V6.1 for i5/OS®
- WebSphere Application Server for Developers V6.1 for i5/OS
- WebSphere Application Server Network Deployment V6.1 for i5/OS
- WebSphere Application Server – Express V6.1 for i5/OS

The home page for WebSphere Application Server on System i can be found at:

http://www-03.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/

Application support

WebSphere Application Server V6.1 can run the following types of applications:

- J2EE applications
- Portlet applications
- Session Initiation Protocol (SIP) applications

J2EE applications

The Java 2 Platform, Enterprise Edition specification is the standard for developing, deploying, and running enterprise applications. WebSphere Application Server V6.1 provides full support for the J2EE 1.4 specification.

The J2EE programming model has four types of application components:

- Enterprise beans
- Servlets and JavaServer™ Pages™ files
- Application clients

The primary development tool for WebSphere Application Server J2EE 1.4 applications is Rational® Application Developer. The Application Server Toolkit, shipped with WebSphere Application Server, also contains the tools needed to create, test, and deploy J2EE 1.4 applications and, in addition, includes full support for the new features of J2SE™ 5.0. Applications are packaged as enterprise application archives (EAR files).

For information about the J2EE specification, see http://java.sun.com.

Portlet applications

The Portlet container in WebSphere Application Server V6.1 provides the runtime environment for JSR 168 compliant portlets.

Portlet applications are intended to be combined with other portlets to collectively create a single page of output. The Portlet container takes the output of one or more Portlets and generates a complete page that can be displayed.

The primary development tool for portlets on WebSphere Application Server portlet applications is the Application Server Toolkit. You can also use Rational Application Developer, but should review the following item in the WebSphere Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/cport_portlets.html

Portlets are packaged in WAR files.

Note that the portlet runtime does not provide the advanced capabilities of WebSphere Portal, such as portlet aggregation and page layout, personalization and member services, or collaboration features.

For more information about JSR 168, see:

http://jcp.org/en/jsr/detail?id=168

Session Initiation Protocol (SIP) applications

SIP applications are Java programs that use at least one Session Initiation Protocol servlet written to the JSR 116 specification. SIP is used to establish, modify, and terminate multimedia IP sessions. SIP negotiates the medium, the transport, and the encoding for the call. After the SIP call has been established, the communication takes place over the specified transport mechanism, independent of SIP. Examples of application types that use SIP include voice over IP, click-to-call, and instant messaging.

The Application Server Toolkit provides special tools for developing SIP applications. SIP applications are packaged as SIP archive (SAR) files and are deployed to the application server using the standard WebSphere Application Server administrative tools. SAR files can also be bundled within a J2EE application archive (EAR file), similar to other J2EE components.

For more information, see:

- JSR 116 SIP Servlet API 1.0 Specification

  http://www.jcp.org/aboutJava/communityprocess/final/jsr116/

- RFT 3261

  http://www.ietf.org/rfc/rfc3261.txt

Application server configurations

At the heart of each member of the WebSphere Application Server family is an application server. Each family has essentially the same architectural structure. Although the application server structure for Base and Express is identical, there are differences in licensing terms, the development tool that is provided, and platform support.

With Base and Express, you are limited to stand-alone application servers. Network Deployment enables more advanced topologies that provide workload management, scalability, high availability, and central management of multiple application servers.

Runtime environments are built by creating profiles. A profile can define a deployment manager, a stand-alone application server, or an empty node to be federated (added) to a cell. Each profile contains files specific to that runtime such as logs and configuration files. Profiles can be created during and after installation. After the profiles have been created, further configuration and administration is performed using the WebSphere administrative tools.

Stand-alone server configuration

Express, Base, and Network Deployment all support a single stand-alone server environment. With a stand-alone configuration, 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. Each stand-alone server is created by defining an application server profile.

Multiple stand-alone application servers can exist on a machine, either through independent installations of the WebSphere Application Server code or through multiple profiles within one installation. However, WebSphere Application Server does not provide for central management or administration for multiple application servers. Stand-alone application servers do not provide workload management or failover capabilities.

Figure 1 on page 6 shows an architectural overview of a stand-alone application server.

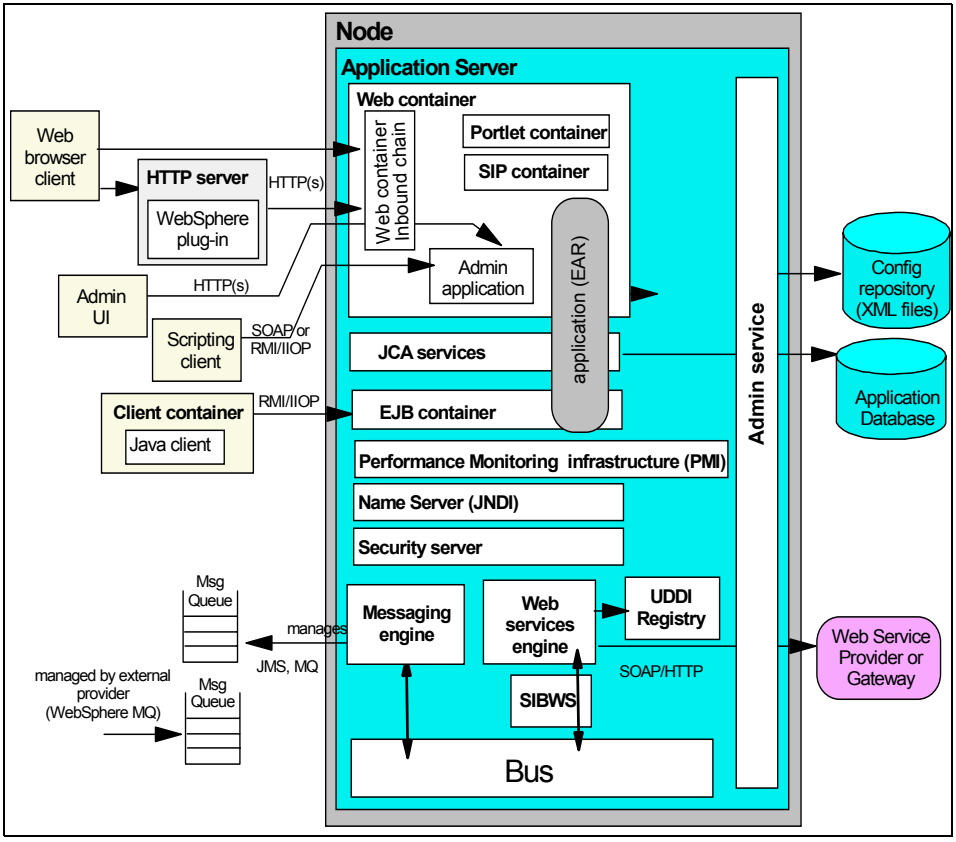

Figure 1 Architectural overview for a stand-alone application server

Distributed server configuration

With Network Deployment, you can build a distributed server configuration, which enables central administration, workload management, and failover. In this environment, you integrate one or more application servers into a cell that is 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 handled centrally from the administration interfaces via the deployment manager.

With a distributed server configuration, you can create multiple application servers to run unique sets of applications and then manage those applications from a central location. However, more important, you can cluster application servers to allow for workload management and failover capabilities. Applications that you install in the cluster are replicated across the application servers. When one server fails, another server in the cluster continues processing. Work is

distributed among Web and Enterprise JavaBeans™ (EJB™) containers in a cluster using a weighted round-robin scheme.

Figure 2 illustrates the basic components of an application server in a distributed server environment.

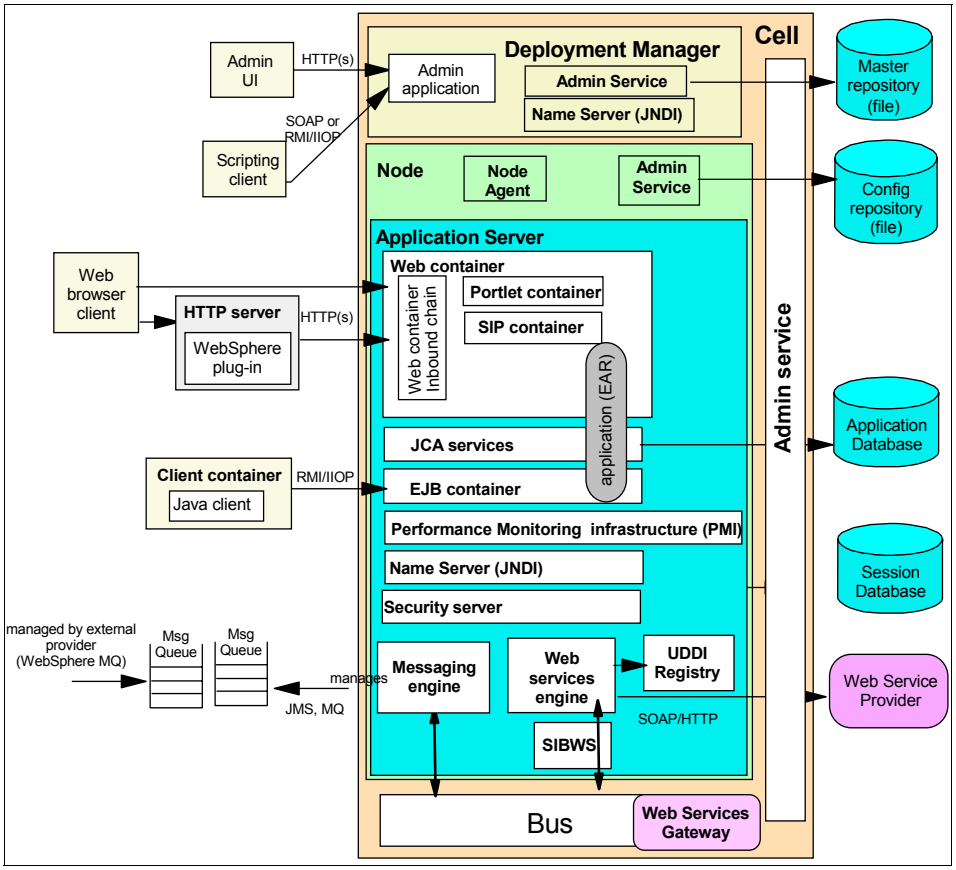

Figure 2 Distributed server environment

A distributed server configuration can be created in one of three ways:

- Create a deployment manager profile to define the deployment manager. Then create one or more custom node profiles to be federated as nodes into the cell managed by the deployment manager. The custom nodes can exist on the deployment manager machine or on multiple separate machines. Application servers can then be created using the administrative tools, for example the administrative console.
- Create a deployment manager profile to define the deployment manager. Then create one or more application server profiles and federate these

profiles into the cell managed by the deployment manager. This process adds both nodes and application servers into the cell. The application server profiles can exist on the deployment manager machine or on multiple separate machines.

- Create a cell profile. This actually creates two profiles: a deployment manager profile and a federated application server profile. Both reside on the same machine.

Application servers, nodes, and cells

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

Application servers

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

Nodes, node groups, and node agents

A node is a grouping of application servers for configuration and operational management on one machine. Nodes are generally associated with a physical machine. It is possible to have multiple nodes on a single machine, but nodes cannot span machines. In a stand-alone application server environment, there is only one node. With Network Deployment, you can configure multiple nodes in a distributed server environment that are managed from one central administration server.

In centralized management configurations, each node has a node agent that works with the deployment manager to manage administration processes. The node agent is created under the covers when you federate a stand-alone node to a cell.

A node group is a grouping of nodes within a cell that have similar capabilities. A node group validates that the node is capable of performing certain functions

before allowing those functions. For example, a cluster cannot contain both z/OS nodes and nodes that are not z/OS. In this case, you can define multiple node groups, one for the z/OS nodes and one for nodes other than z/OS. A group called the DefaultNodeGroup is automatically created based on the deployment manager platform. This node group contains the deployment manager and any new nodes with the same platform type. A node can be a member of more than one node group.

On the z/OS platform, a 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.

Cells

A cell is a grouping of nodes into a single administrative domain. In the Base and Express configurations, a cell contains one node and that node contains one application server.

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

It is possible to have a cell comprised of nodes on mixed platforms. This is referred to as a heterogeneous cell.

Servers

WebSphere Application Server supplies application servers, which provide the functions that are required to host applications and proxy servers that distribute work to the application servers. It also provides the ability to define external servers to the administration process.

Table 1 shows which types of servers you can define.

Table 1 WebSphere Application Server server support

	Express and Base	**Network Deployment**
Application server	Yes	Yes
Application server cluster	No	Yes
External Web server	Yes	Yes

	Express and Base	Network Deployment
External generic server	No	Yes
Generic server cluster	No	Yes
Proxy server	No	Yes
WebSphere MQ Server	Yes	Yes
WebSphere V5 Java Message Server servers	No	Yes

Application servers

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™).

WebSphere Application Server V6.1 has a new JVM designed to improve stability and performance. It provides a Java language compiler and execution environment to support the Java 2 Standard Edition (J2SE) 5 specification. This new JVM is supported on all platforms that ship with an IBM JDK™. (J2SE 5 is currently not used on Solaris, HP, and i5/OS.)

Application server clusters

With Network Deployment, you can use application server clustering to enhance 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 all have identical application components deployed on them. Other than the applications that are 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 is 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. A cluster can span machine or LPAR boundaries and can span across operating

systems with one exception. A cluster cannot span z/OS and non-z/OS platforms.

When you install, update, or delete an application, the updates are automatically distributed to all members in the cluster. A rollout update option enables you to update and restart the application servers on each node, one node at a time, providing continuous availability of the application.

Proxy server

A proxy server is a specific type of application server that routes HTTP requests to application servers that host the applications. The proxy server is the initial point of entry, after the firewall, for requests into the enterprise. The proxy server can be configured with rules to route to and load balance the clusters of application servers. The proxy server is also capable of securing the transport, using Secure Sockets Layer (SSL), and the content using various authentication and authorization schemes. Another important feature is its capability to protect the identity of the content servers from the Web clients by using response transformations (URL rewriting). The proxy server can also improve performance by caching content locally and by protecting the content servers from surges in traffic.

WebSphere MQ server

A WebSphere MQ server defines the location and attributes of a z/OS WebSphere MQ queue manager or queue sharing group. It is designed to take advantage of the load balancing and availability features of WebSphere MQ on z/OS.

Java Message Server servers (V5)

In WebSphere Application Server V5, Java Message Server servers provide 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, Network Deployment supports existing Java Message Server servers in V5 application servers in the cell.

External servers

You can define servers other than WebSphere application servers to the administrative process.

Generic servers

A generic server is a server that is managed in the WebSphere administrative domain but is not supplied by the WebSphere Application Server product. 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 server.

You can also create a cluster of generic servers and use a proxy server to route requests to members of the cluster.

Web servers

Web servers can be defined to the administration process as a Web server node, enabling applications to be associated 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 that enables the deployment manager to administer the Web server. You can start or stop the Web server from the deployment manager, generate the Web server plug-in for the node, and automatically push it to the Web server. You would normally have managed Web server nodes behind the firewall with the WebSphere Application Server installations.

Unmanaged Web server nodes, as the name implies, are not managed by WebSphere. You would normally find these outside the firewall or in the demilitarized zone. You must manually copy or FTP Web server plug-in configuration files to the Web server. However, if you define the Web server as a node, you can generate custom plug-in configuration files for it. In a z/OS environment, you must use unmanaged nodes if the Web server is a non-z/OS product.

Note: 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. Then, you can automatically push the plug-in configuration file to the Web server with the deployment manager using HTTP commands to the IBM HTTP Server administration process. This configuration does not require a node agent.

IBM HTTP Server is shipped with all WebSphere Application Server packages.

Web server plug-ins

A Web server can serve requests that do not require any dynamic content (for example, HTML pages). However, when a request requires dynamic content, such as JavaServer Pages (JSP™) or servlet processing, it must be forwarded to WebSphere Application Server for handling.

To forward a request, you use a Web server plug-in that is included with the WebSphere Application Server packages for installation on a Web server. You copy an Extensible Markup Language (XML) configuration file, configured on the WebSphere Application Server, 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 WebSphere Application Server receives a request for an application server, it forwards the request to the appropriate Web container in the application server. The plug-in can use HTTP or HTTPs to transmit the request.

Containers

Containers provide runtime support for applications. WebSphere Application Server V6.1 has the following container support.

Application server containers

Each application server provides the following container support:

- Web container

  The Web container processes servlets, JSPs (processed as servlets), 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.

  Requests are received by the Web container through 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. Requests for HTML and other static content that are directed to the Web container are served by the Web container inbound chain.

  Although the Web container can serve static content, a more likely scenario is that you will use an external Web server to receive client requests and a Web server plug-in to forward requests for servlets to the Web container.

- EJB container

  The Enterprise JavaBeans container provides all of 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 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 enterprise beans. A single container can host more than one EJB Java archive (JAR) file.

- Portlet container

  The portlet container processes JSR168 compliant portlets. The portlet container is an extension to the Web container.

- Session Initiation Protocol container

  The SIP container processes applications that use at least one SIP servlet written to the JSR 116 specification. The portlet container is an extension to the Web container.

Application client container

The application client container is a separately installed component on the client's machine. It enables the client to run applications in a J2EE environment that is compatible with EJB.

Application server services

The application server also provides the following basic services to support application processing.These services will be discussed in the following sections.

- J2EE Connector Architecture services
- Transaction service
- Dynamic cache service
- Message listener service
- Object Request Broker service
- Administrative service (Java Management Extensions)
- Diagnostic trace service
- Debugging service
- Name service (Java Naming Directory Interface)
- Performance Monitoring Interface service
- Security service (JAAS and Java 2 security)
- Service Integration Bus service

In addition to these services, the following services are provided to support the Programming Model Extensions:

- Application profiling service
- Compensation service
- Internationalization service
- Object pool service
- Startup beans service
- Activity session service
- Work area partition service
- Work area service

J2EE Connector Architecture services

Connection management for access to enterprise information systems (EIS) in WebSphere Application Server is based on the J2EE Connector Architecture (JCA) specification, also sometimes referred to as J2C. 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.

Within the application server, the Connection Manager pools and manages connections. The Connection Manager administers connections that are obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC™ 2.0 Extensions (and later) specification.

Transaction service

WebSphere applications 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 the XAResource interface and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service protocol (for example, application servers) or the Web Service Atomic Transaction protocol.

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

How applications use transactions depends on the type of application component; for example:

- 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.

WebSphere Application Server handles transactions with 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 (such as databases). Optionally, the container can control the demarcation of transactions for enterprise beans that are configured for container-managed transactions.
- An API (UserTransaction) that is available to bean-managed enterprise beans and servlets that enables such application components to control the demarcation of their own transactions.

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 objects that can be cached (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 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 then copied or replicated to other servers in the cluster, 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, enabling 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 Edge Side Include (ESI) processor. The ESI processor caches whole pages, as well as fragments, providing a higher cache hit ratio. The cache that is 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 controls caches outside of the application server, such as that provided by the Edge components, an IBM HTTP Server's FRCA cache that is not z/OS, and a WebSphere HTTP Server plug-in ESI Fragment Processor that is not z/OS. When external cache groups are defined, the dynamic cache matches external cache entries with those groups and pushes out cache entries and invalidations to those groups. This external caching enables 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 performance.

Message listener service

With EJB 2.1, an ActivationSpec is used to connect message-driven beans to destinations. However, you can deploy existing EJB 2.0 message-driven beans against a listener port as in WebSphere V5. For those message-driven beans, 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 message-driven bean.

Object Request Broker service

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

The ORB service 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. The ORB service provides 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 parameters, 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, which returns the result 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.

Administrative service

The administrative service runs within each server JVM. In Base and Express, the administrative service runs in the application server. In Network Deployment, 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 functionality that provides different levels of administration to certain users or groups using the following administrative roles:

- Administrator
- Monitor
- Configurator
- Operator

This administrative security can be enabled during installation and profile creation. If you choose this option, a federated security repository will be created containing a single file-based user registry in it.

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 hosted by the application server. 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 application developers use. 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 containing 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.

WebSphere Application Server name space features include:

- 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 that is used for cell-scoped persistent bindings and a node persistent root that is used to bind objects with a node scope.

- Federated name space structure

  A name space is a collection of all names that are 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 to cooperatively form 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 a 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

  You can use the configuration graphical interface and script interfaces 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 (INS) object Uniform Resource Locator (URL)

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

Figure 3 summarizes the naming architecture and its components.

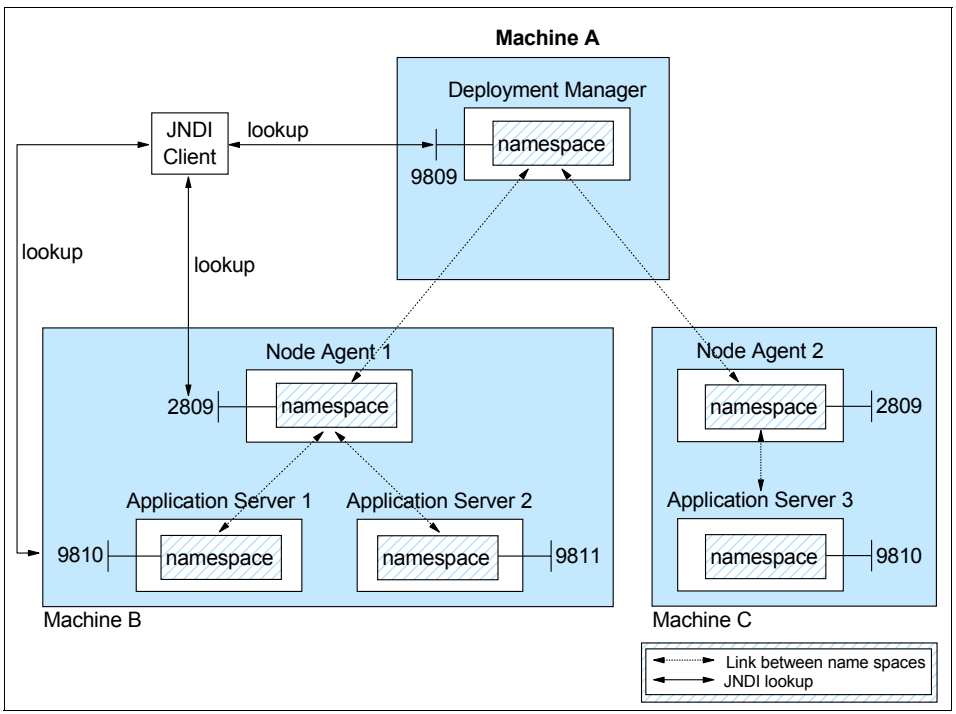

Figure 3 Naming topology

Performance Monitoring Infrastructure 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 Java Management Extensions (JMX™) client. WebSphere Application Server contains Tivoli® Performance Viewer, which is integrated in the WebSphere administrative console and 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 the 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 agents that Tivoli monitoring tools use.

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.

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 Simple Object Access Protocol (SOAP).

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 V6.0)
- Dynamic cache replication

Replication is done through the use of replication domains that consist 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, you need a separate domain for dynamic cache replication.

You can define a domain so that each domain member has a single replicator that sends data to another member of the domain, or so that each member has multiple replicators that send data to multiple members of the domain.

WebSphere Application Server offers two topologies when setting up data replication among servers:

- Peer-to-peer topology

  Each application server stores sessions in its own memory 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 user 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.

Virtual hosts

A virtual host is a configuration that enables a single host machine to resemble multiple host machines. This configuration enables a single physical machine to support several independently configured and administered applications. A virtual host is not associated with a particular node. It is a configuration, rather than a live object, which is why you can create it but you cannot start or stop it.

Each virtual host has a logical name and a list of one or more Domain Name Server (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 that are 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

  This virtual host is 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

  This virtual host is configured specifically 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`

Session management

In many Web applications, users dynamically collect data as they move through the site based on a series of selections on the pages that they visit. Where the user goes and what the application displays might depend on what the user has chosen previously from the site. To maintain this data, the application stores it in a *session*.

WebSphere supports three approaches to track sessions:

- Secure Sockets Layer (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 2 shows the session support for each WebSphere Application Server configuration.

Table 2 WebSphere Application Server session management support

	Express and Base	**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 be accessed only by a single Web application. However, there might be a logical reason for multiple Web applications to share information (for example, sharing a user name). WebSphere Application Server provides an IBM extension to the specification that enables session information to be shared among Web applications within an enterprise application. This option is offered as an extension to the application deployment descriptor. No code change is necessary to enable this option. You specify this option during application assembling.

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, which is 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 obtain their session information only if they return to the machine and WebSphere Application Server holds their session information about subsequent accesses to the Web site.

Most important, local session management lacks a persistent store for the sessions it manages. A server failure takes down not only the WebSphere instances that are running on the server but also destroys any sessions that are 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 option instructs WebSphere to place session objects in a persistent store. Using a persistent store enables an application server to recover the user session data 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 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.

Stateful session EJB persistence

With WebSphere Application Server V6, you now have failover capability of stateful session EJBs. This function uses data replication services and interacts with the workload manager component during a failover situation.

Web services

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. WebSphere Application Server supports SOAP-based Web service hosting and invocation.

WebSphere Application Server can act as both a Web service provider and as a requester.

As a requester, WebSphere Application Server hosts applications that invoke Web services from other locations.

As a provider, WebSphere Application Server hosts Web services that are published for use by clients. 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 automatically added, and a SOAP admin tool is included in a Web module. If you are not using Rational Application Developer or another IBM development tool that provides this function, you can use the endptEnabler tool found in the WebSphere bin directory to enable the SOAP services within the Enterprise Application Archive (EAR) file and to add the SOAP admin tool.

Web services support includes:

- Universal Discovery Description and Integration (UDDI)

  A global platform-independent, open framework that enables 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.

- Java API for XML-based RPC (JAX-RPC) (JSR 101)

  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 it supports JavaBeans and enterprise beans as Web service providers.

- Web Services for J2EE specification (JSR 109)

  This specification adds EJBs and XML deployment descriptors to JSR 101.

- WS-Security

  This specification 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.

- Java API for XML Registries (JAXR)

  JAXR is an API that standardizes access to Web services registries from within Java. JAXR 1.0 defines access to ebXML and UDDI V2 registries. WebSphere Application Server provides JAXR level 0 support, meaning that 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.

- Simple Object Access Protocol (SOAP)

  A lightweight protocol for exchange of information in a decentralized, distributed environment.

- SOAP with Attachments API for Java (SAAJ)

  A standard for sending XML documents over the Internet from the Java platform.

- WS-I Basic Security Profile

  WS-I BSP promotes interoperability by providing clarifications and amplifications to a set of non-proprietary Web services specifications. WebSphere Application Server Web Services Security provides configuration options to ensure that the BSP recommendations and security considerations can be enabled to ensure interoperability.

- WS Resource Framework

  A generic framework for modelling and accessing stateful resources using Web services, so that the definition and implementation of a service and the integration and management of multiple services is made easier.

- WS Transaction support

  Defines how Web services applications can work within global transactions in enterprise environments using the following three specifications:

  - WS-Atomic Transaction (WS-AT)

    A specific coordination type that defines protocols for atomic transactions.

  - WS-Business Activity (WS-BA)

    A specific coordination type that defines protocols for business activities. A business activity is a group of general tasks that you want to link together so that the tasks have an agreed outcome.

  - WS-Coordination (WS-Coor)

    Specifies a context and a registration service with which participant Web services can enlist to take part in the protocols that are offered by specific coordination types.

- WS-I Basic Profile

  A set of non-proprietary Web services specifications that promote interoperability.

- WS- Notification

  Publish and subscribe messaging for Web services.

- WS-Addressing

  Enables systems to support message transmission and identification through networks that include firewalls or gateways in a transport-neutral manner.

- Implementing Enterprise Web Services - JSR 109 and JSR 921

  Defines the programming model and run-time architecture to deploy and look up Web services in the J2EE environment; more specifically, in the Web, EJB, and Client Application containers. One of its main goals is to ensure vendors' implementations interoperate.

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

- Custom bindings

  JAX-RPC does not support all XML schema types. Custom bindings enable developers to map Java to XML and XML to Java conversions.

- Support for generic SOAP elements

  In cases where you might want generic mapping, this support enables you to eliminate binding and use the generic SOAPElement type.

- Multi-protocol support

  This features allows a stateless session EJB as the Web service provider, which provides enhanced performance without changes to the JAX-RPC client.

- 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 this server-side caching, WebSphere Application Server V6 introduces caching for Web services clients running within a Version 6 application server, including the Web Services Gateway.

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 Web Services Invocation Framework (WSIF) to expose an implementation as a service.

IBM WebSphere UDDI Registry

WebSphere Application Server V6 provides a private UDDI registry that implements Version 3.0 of the UDDI specification. This registry enables the enterprise to run its own Web services broker within the company or to provide

brokering services to the outside world. The UDDI registry installation and management is now integrated with WebSphere Application Server.

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.

Access to the registry for inquiry and publish can be done through:

- The UDDI SOAP API.
- The UDDI EJB client interface.
- The UDDI user console. You can use this Web-based graphical user interface to publish and to inquire about UDDI entities. However, it provides only 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.

Web Services Gateway (Network Deployment only)

The Web Services Gateway bridges the gap between Internet and intranet environments during Web service invocations. The gateway builds upon the WSDL and the WSIF for deployment and invocation.

With WebSphere Application Server 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 of the service integration bus. Handlers monitor messages at ports and take appropriate action, depending on 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. However, the service endpoint is changed to the gateway, which is now the official endpoint for the service client. Figure 4 diagrams the configuration for exposing Web services through a gateway.

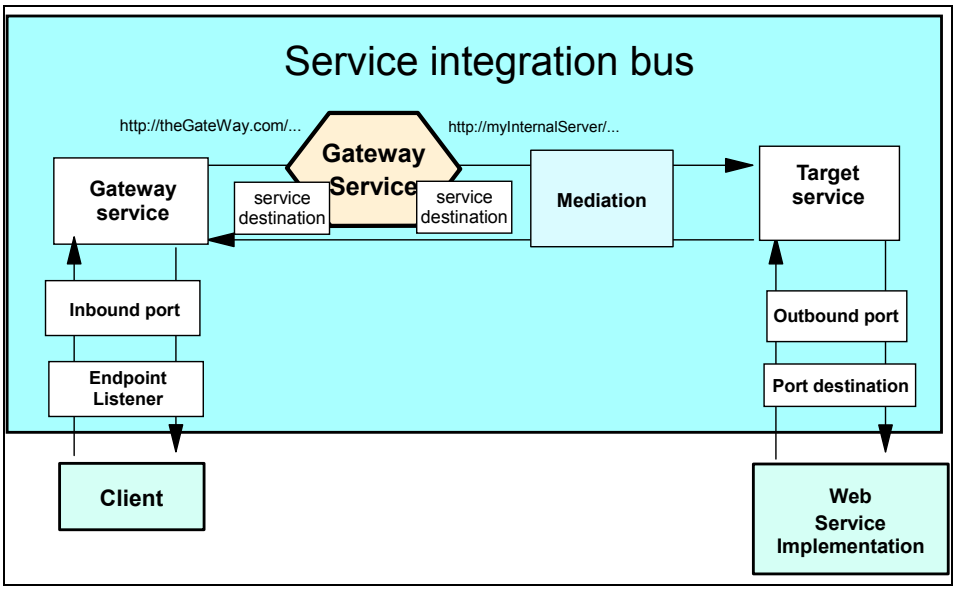

Figure 4 Exposing Web services through a gateway

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 allows internal service

requestors to invoke the service as though it were running on the gateway. Again, a new WSDL is generated by the gateway that shows the same interface but that names 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 that shown in Figure 4 on page 30, 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.

Service integration

Service integration technology provides the communication infrastructure for messaging and service-oriented applications, thus unifying this support into a common component. Service integration includes:

- A JMS 1.1 compliant JMS provider. This provider is referred to as the default messaging provider.
- The service integration bus (referred to as the *bus*). The service integration bus provides the communication infrastructure for the default messaging provider. The bus supports the attachment of Web services requestors and providers. The bus also provides the means to implement intermediary logic (mediations) to intelligently adapt message flow in the network.
- Support for the Web services gateway, which provides you with a single point of control, access and validation of Web service requests, and enables you to control which Web services are available to different groups of Web service users.

Service integration bus

Service integration bus capabilities are fully integrated into WebSphere Application Server, enabling it to take advantage of WebSphere security, administration, performance monitoring, trace capabilities, and problem determination tools.

Figure 5 illustrates the service integration bus and how it fits into the larger picture of an enterprise service bus.

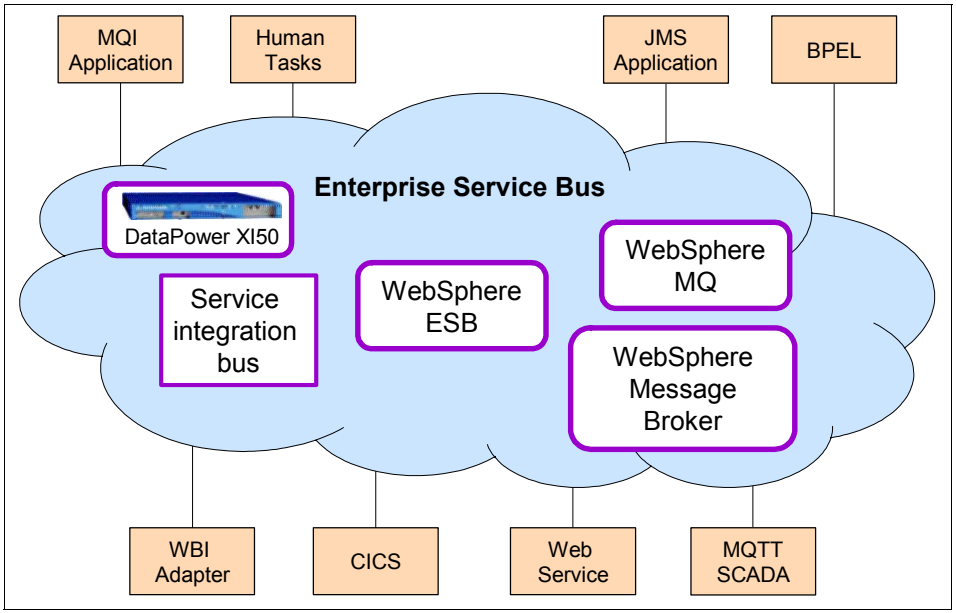

Figure 5 The enterprise service bus

A service integration bus consists of:

- 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 created automatically on the application server or cluster. The messaging engine provides a connection point for clients to produce or from where to consume messages.

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

- Destinations

  The place within the bus to which 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

  A messaging engine uses a message store to save information that is needed for recovery in the event of a failure, including messages, subscription information, and transaction states. Each messaging engine has one and only one message store. This can be either a file store or a data store.

  With a file store (the default), information is stored in a file system via the operating system.

  With a data store, information is stored in a relational database. Multiple messaging engines can share a database for the data store, each with its own set of tables and schema.

The service integration bus supports the following application attachments:

- Messaging applications
  - JMS applications running in either WebSphere Application Server V5 or WebSphere Application Server V6 can connect to the bus using the JMS programming model. WebSphere Application Server V5 clients make use of the default V5 messaging provider and require the bus to be configured with an MQ client link. WebSphere Application Server V6 clients use the default messaging provider.
- 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

Service integration bus and messaging

With Express or Base, you typically have one stand-alone server with one messaging engine on one service integration bus. With Network Deployment, you have more flexibility to use multiple buses for high availability and scalability.

Figure 6 illustrates two application servers, each with a messaging engine on a service integration bus.

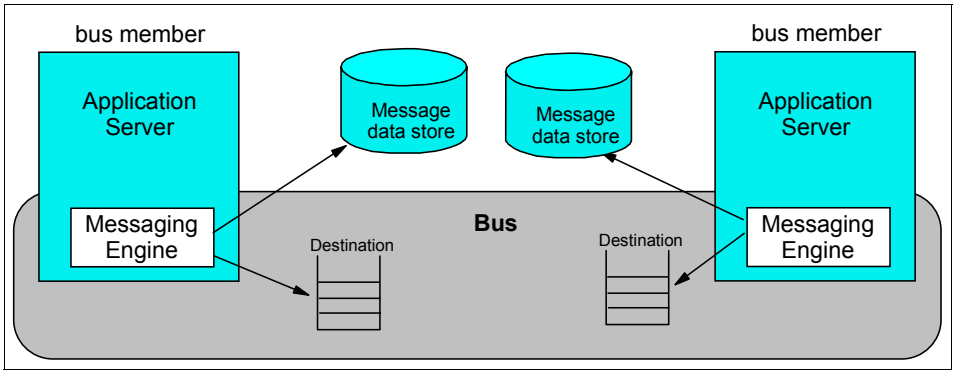

Figure 6 Service integration bus

The following topologies are valid:

- One bus and one messaging engine (application server or cluster).
- One bus with multiple messaging engines.
- Multiple buses within a cell that may or may not be connected to each other.
- Buses connected between cells.
- One application server that is a member of multiple buses and that has one messaging engine per bus.
- A connection between a bus and a WebSphere MQ queue manager. When using this type of topology, you should consider the following points:
  - A messaging engine cannot participate in a WebSphere MQ cluster.
  - You can configure the messaging engine to look like another queue manager to WebSphere MQ.
  - WebSphere applications can send messages directly to WebSphere MQ or though the service integration bus.
  - JMS clients running in WebSphere Application Server V5, or the V5 application client, can connect to the bus using the default V5 messaging provider (previously named WebSphere JMS Provider). A JMS client running in WebSphere Application Server V6, or the V6 application client, can also connect to the Version 5 JMS server using the default V5 messaging provider.

Clustering

In a distributed server environment, you can use clustering for high availability and scalability. You can add a cluster as a bus member and achieve:

- 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

You can define quality of service on a destination basis to determine how messages are (or are not) persisted. You can also specify quality of service within the application.

Message driven beans

With EJB 2.1, message driven beans (MDB) 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).

Web services and the bus

Web services support on the bus is provided by the Web services enablement (SIBWS) component. This support is installed by the administrator after installation and profile creation. SIBWS enables you to:

- Define a service running locally to the bus as a destination and make it available as a Web service.
- Define an external Web service to the bus and make it available internally at a bus destination.
- Use the Web services gateway to map an existing local or external service to a new Web service that seems to be provided by the gateway.

Mediations

A mediation manipulates a message as it traverses the messaging bus (destination). For example, a mediation can:

- Transform the message.
- Reroute the message.
- Copy and route the message to additional destinations.

- Interact with non-messaging resource managers (for example, databases).

You control a mediation using a mediation handler list. The list is a collection of Java programs that perform the function of a mediation that are invoked in sequence.

Bus security

You have the option of enabling bus security to ensure the following security measures on the bus. In order to enable bus security, you must have administrative security enabled.

- Authentication of users when connecting to a bus. Authentication is performed using the user registry that is specified in the WebSphere Application Server global security settings.
- Role-based authorization of users when connecting to a bus and when using the bus resources. Users and groups can be authorized to connect to the bus, to access specific destinations on the bus, and to access topic spaces and topics.
- Establishment of trust between peer messaging engines in a bus. This requires that an authentication alias be set on the bus configuration. When a messaging engine connects to another messaging engine the user ID and password specified by the authentication alias are sent to the peer messaging engine. If these credentials can be authenticated against the cells User Registry, and match the local messaging engines configuration, trust is established and the messaging engines will communicate.
- Authentication of the messaging engine to the database backing the data store (this is supported without first enabling bus security).

You can also secure communication transports between the client and messaging engine and between messaging engines. Secure transport can be configured regardless of the bus security setting.

Security

WebSphere Application Server provides the following support for security:

- Java 2 security
- J2EE security (role mapping)
- JAAS
- CSIv2
- Authentication using SWAM (available in Express and Base only) or Lightweight Third Party Authentication (LTPA)

- User registry: file-based, local OS, LDAP, custom registry, federated repository

Figure 7 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 you can configure according to the requirements and existing IT resources.

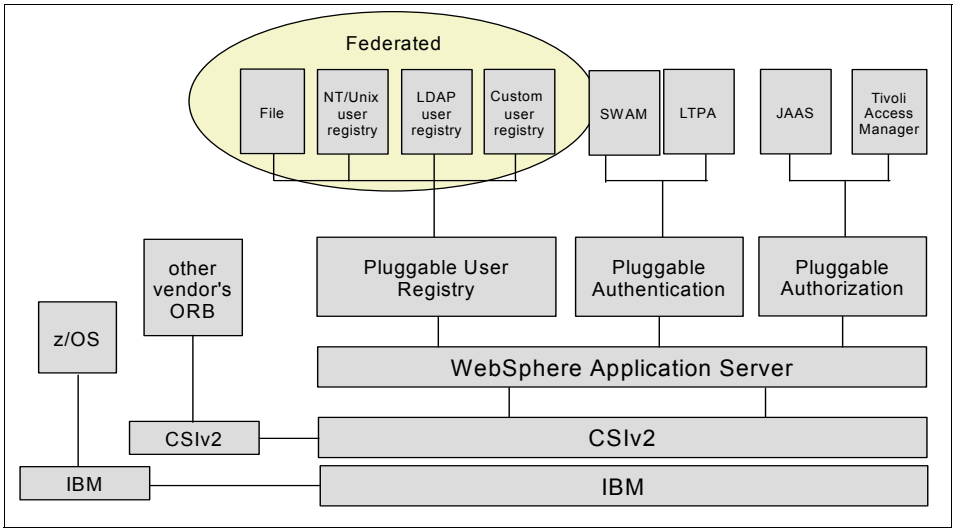

Figure 7 WebSphere Application Server security architecture

WebSphere Application Server security sits on top of the operating system security and the security features provided by other components, including the Java language. This architecture provides the following layers of security:

- 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 WebSphere and the Java security classes use.
- 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. You can activate or inactivate Java 2 security independent 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 Java Authentication and Authorization Services (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 enables 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 security policies based on J2EE and to support J2EE security APIs. WebSphere applications use security APIs to access the security mechanisms and implement security policies. J2EE security guards access to Web resources such as servlets/JSPs and EJB methods based on roles that the application developer defines. Users and groups are assigned to these roles during application deployment.
- Java Contract for Containers (JACC) support allows the use of third-party authorization providers for access decisions. The default JACC provider for WebSphere Application Server is the Tivoli Access Manager that is bundled with Network Deployment. The Tivoli Access Manager client functions are integrated in WebSphere Application Server.
- IBM Java Secure Socket Extension is the SSL implementation that WebSphere Application Server uses. It is a set of Java packages that enable secure Internet communications. It implements a Java version of SSL and Transport Layer Security 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 policies in a unified manner for both Web and EJB resources. WebSphere global security options are defined at the cell level. However, individual servers can override a subset of the security configuration. When using mixed z/OS and distributed nodes, the security domain features are merged.

User registry

The pluggable user registry enables you to configure different databases for storing user IDs and passwords that are used for authentication and authorization. Only one single registry can be active at a time.

These are your options for user registries:

- File-based repository

  New with V6.1, this repository can be managed through the administrative interfaces including the administrative console. A new WebSphere component called virtual member manager (VMM) provides the interface to the file-based repository.

- Local operating system user registry

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

- LDAP user registry

  An LDAP user registry is often the best solution for large-scale Web implementations. Most LDAP servers on the market are well equipped with security mechanisms that you can use to securely communicate with WebSphere Application Server. WebSphere offers considerable flexibility in setting search parameters to adapt to different LDAP schemas.

- Custom user registry

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

- Federated repository

  The federated repository option enables you to use multiple registry types together to form a single logical repository. With the federated repository, you can add attributes to identities without affecting the underlying registry.

  You can federate any of the registry types together, although configuring federated repository types from the administrative console is limited to LDAP and file-based registries. To configure a federated repository with custom or local operating system you must use wsadmin commands or VMM APIs. When it is configured, the administrative console can be used to manage entries in one selected registry.

Authentication

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either a user, a machine, or an application. The pluggable authentication module enables 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 that differ primarily in the distributed security features each supports:

- Simple WebSphere Authentication Mechanism is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction exists because this mechanism does not support forwardable credentials. So, 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. Instead, an unauthenticated credential is transmitted that, depending on the security permissions configured on the EJB methods, might cause authorization failures.

  Because the Simple WebSphere Authentication Mechanism is intended for a single application server process, single sign-on is not supported.

  This type of authentication is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

  Simple WebSphere Authentication Mechanism relies on the session ID and is not as secure as Lightweight Third Party Authentication (LTPA). Thus, we strongly recommend using SSL with this type of authentication.

- Lightweight Third Party Authentication is intended for distributed, multiple application servers and machine environments. It supports forwardable credentials and single sign-on. LTPA can support security in a distributed environment through the use of cryptography, which enables it to encrypt,

digitally sign, and securely transmit authentication-related data and later decrypt and verify the signature.

This type of authentication requires that the configured user registry be a central, shared repository such as LDAP or a Windows domain type registry.

Authorization

WebSphere Application Server standard authorization features are as follows:

- Java 2 security architecture, which uses a security policy to specify who is allowed to execute code in the application. Code characteristics, such as a code signature, signer ID, or source server, determine whether the code is granted access to be executed.
- JAAS, which extends the Java 2 approach with role-based access control. Permission to execute a code is granted based on the code characteristics and on the user running it. JAAS programming models enable 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 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 that are defined by JACC. Subsequent authorization decisions are made using this policy information. An exception is that the WebSphere Application Server default authorization engine makes all administrative security authorization decisions.

Security components

Figure 8 on page 42 shows an overview of the security components that come into play in WebSphere Application Security.

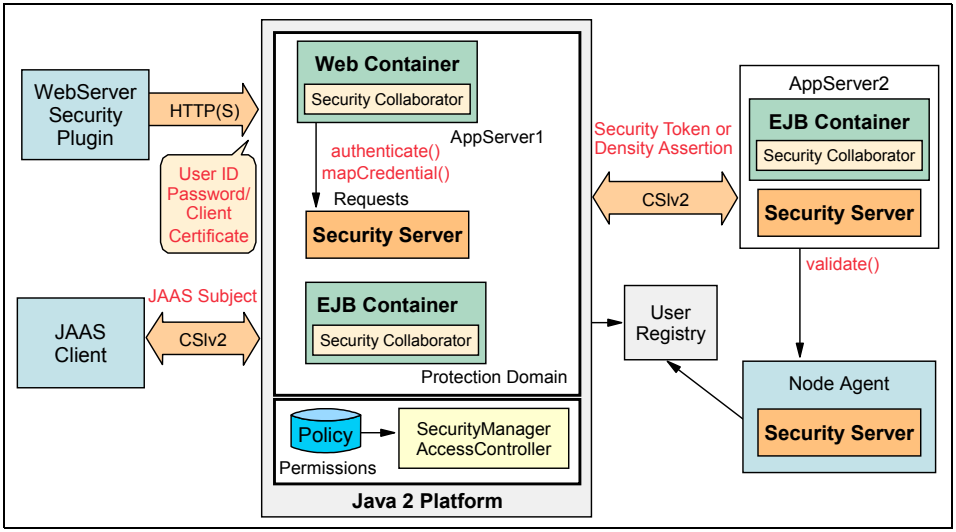

Figure 8 WebSphere Application Security components

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 the deployment descriptors specify. These processes communicate with the security server every time authentication and authorization actions are required. The following security collaborators are identified:

- Web security collaborator

  Resides in the Web container and provides the following services to the application:

  - Checks authentication
  - Performs authorization according to the constraint specified in the deployment descriptor
  - Logs security tracing information

- EJB security collaborator

  Resides in the EJB container and uses Common Secure Interoperability Version 2 (CSIv2) and Secure Authentication Service (SAS) to authenticate Java client requests to enterprise beans. The EJB security collaborator 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 external ORB using CSIv2 when a request for a remote bean is issued

Security flows

The following sections outline the general security flow.

Web browser communication

When a Web browser sends a request to a WebSphere application, the following interactions occur from a security point of view:

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.
3. Using the Web server plug-in, the Web server redirects the request 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, and the following 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 the user registry for authentication and authorization.

Java client communication

When a Java client interacts with a WebSphere application, the following occurs:

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 reads the user credentials 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 user to the requested resource.

Resource providers

Resource providers define resources that running J2EE applications need. WebSphere Application Server provides support for the following resource providers:

- JDBC provider
- Mail providers (JavaMail™)
- JMS providers
- Resource environment providers
- URL providers
- Resource adapters

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 technique makes an application more portable, because the application does not need to hard code a driver name, which often includes the name of a particular vendor. The technique also makes maintaining the code easier. If, for example, you move the data source to a different server, all you have to do is update the relevant property in the data source. You do not have to touch the code using that data source.

After 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 is usually 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, you are providing information about the set of classes that is used to implement the data source and the database driver. That is, the JDBC provider holds 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, as shown in Figure 9 on page 46.

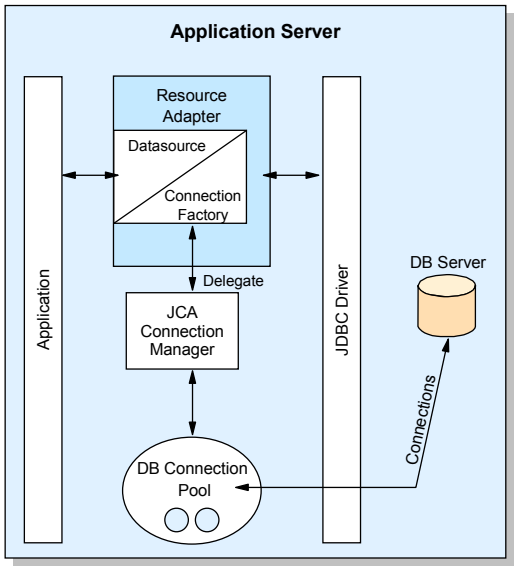

Figure 9 Resource adapter in J2EE connector architecture

The JCA Connection Manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides the JDBC wrappers and JCA CCI implementation that allow applications using bean-managed persistence, JDBC calls, and container-managed persistence beans to access the database JDBC Driver.

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 4.0.

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 the following protocol providers:

- Simple Mail Transfer Protocol (SMTP)

  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)

  The standard protocol for receiving mail.

- Internet Message Access Protocol (IMAP)

  An alternative protocol to POP3 for receiving mail.

These protocol providers are installed as the default and should be sufficient for most applications. 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), 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, which contains the JavaMail APIs as well as the SMTP, IMAP, and POP3 service providers.
- activation.jar, which contains the JAF.

JCA resource adapters

The JCA defines a standard architecture for connecting the J2EE platform to heterogeneous EIS (for example, ERP, mainframe transaction processing, database systems, and legacy applications 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, 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 container-managed persistence beans.

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.

The initial WebSphere configuration includes Default URL Provider. This provider uses the URL support that the IBM JDK provides. 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 the JDK does not support.

JMS providers

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

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

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

The support provided by WebSphere administration tools for configuration of JMS providers differs depending on the provider. Table 3 on page 49 provides a summary of the support.

Table 3 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

WebSphere Application Server provides a default JMS messaging provider. The underlying messaging transport for this provider is implemented using the service integration bus, which provides point-to-point as well as publish and subscribe functions. Connection factories and JMS destinations that are defined to the default messaging provider correspond to destinations on the bus.

WebSphere MQ messaging 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 messaging providers

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

V5 default messaging provider

For backward 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. You can also use the V5 default messaging provider with a service integration bus.

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 that are defined in an application's standard deployment descriptors. The J2EE specification separates the definition of the resource environment entry from the application by:

- 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).
- 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:

- Resource environment provider

  Defines an administrative object that groups together the referenceable, resource environment entry administrative objects and 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.

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 as shown in Figure 10 on page 51.

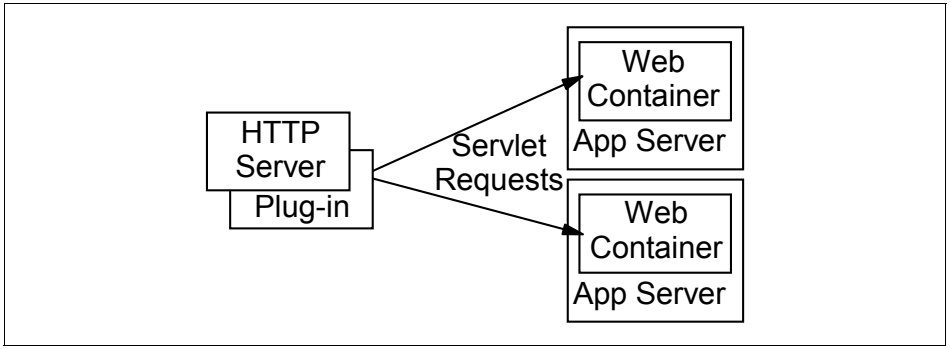

Figure 10 Plug-in (Web container) workload management

This routing is based on weights that are associated with the cluster members. If all cluster members have identical weights, the plug-in sends equal requests to all members of the cluster, assuming no strong affinity configurations. If the weights are scaled in the range from zero to 20, the plug-in routes requests to those cluster members with the higher weight value more often. No requests are sent to cluster members with a weight of zero unless no other servers are available. Weights can be changed dynamically during runtime by the administrator.

A general 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 as shown in Figure 11 on page 52.

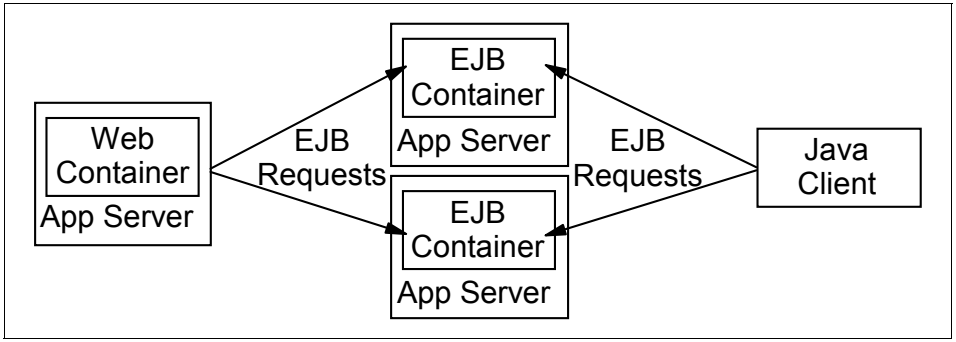

Figure 11 EJB workload management

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 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 is that all servers receive the same number of requests. If the weights for the servers are not equal, the distribution mechanism sends more requests to the higher weight value servers than the lower weight value servers. The policy ensures the desired distribution, based on the weights that are assigned to the cluster members.

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 are chosen (using the round-robin weight method). Cluster members on remote nodes are chosen only if a local server is not available.

High availability

With Network Deployment V6, the high availability features are significantly improved. This is a quick overview of the failover capabilities:

- Web 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 Web Server failover.

- Web container failover

  The Web 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 is provided. The loss of the deployment manager only affects configuration. We recommend 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 (such as workload management routing, PMI aggregation, JMS messaging, transaction manager, and so on) is provided through the use of high availability domains.

  A high availability domain defines a set of WebSphere processes (a core group) that provides 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 a high availability coordinator, managing the HA activities within the core group processes. If a high availability coordinator server fails, another server in the core group takes over the duties of that coordinator. High availability policies define how the failover occurs.

  Workload management information is shared between core members and failover of critical services is done among them in a 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.

- JMS messaging failover

  The messaging engine keeps messages in a remote data store. 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.

Administration

WebSphere Application Server's administration model is based on the JMX framework. JMX enables 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 that are stored in the server's file system.

The JMX implementation in WebSphere Application Server V6.1 is based on JSR168 and supports JSR160. Previously in 6.0.x, the JMX implementation was based on MX4J. The new JMX implementation supports interoperability between V6.1 and back-level nodes to support mixed-version cells.

Administration tools

Table 4 shows the administration tools that WebSphere Application Server supports by configuration.

Table 4 WebSphere Application Server administration tool support

	Express and Base	**Network Deployment**
Administrative console	Yes	Yes
Commands	Yes	Yes
Scripting client, wsadmin	Yes	Yes
Thin client	Yes	Yes

Administrative console

The administrative console is a Web-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 that is 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 that the deployment manager maintains.

In V6.1, many aspects of configuration with the administrative console have been streamlined in order to reduce the number of “clicks” it takes to reach commonly used panels. A new command assist function has been added to allow administrators to view wsadmin scripting commands (in Jython) for the last action run in the console. Though not currently available for all commands, this function can assist administrators that build wsadmin scripts for commonly used commands. The script commands can be logged and JMX notifications sent to the Application Server Toolkit.

Commands

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

Scripting client

The wsadmin scripting client provides extra flexibility over the Web-based administration application, enabling administration to use the command-line interface. Using the scripting client makes administration quicker and automates the administration of multiple application servers and nodes using scripts.

The scripting client uses the Bean Scripting Framework, which enables you to use a variety of scripting languages for configuration and control. WebSphere Application Server V6 supports two languages: Java Command Language (Jacl) and Jython (or jpython). Jython is the strategic direction, and new tools in WebSphere Application Server V6.1 and the Application Server Toolkit have been provided to assist in building Jython scripts. A Jacl-to-Jython migration assistance tool is also included.

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.

Thin administrative client

The thin administrative client is a lightweight runtime package that enables you to run the wsadmin tool or a standalone administrative Java program remotely.

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.

Centralized administration

The Network Deployment package enables multiple servers and nodes to be administered from a central location. This centralized administration uses 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, maintains the configuration for the servers in the node. Table 5 on page 57 shows the distributed administration that WebSphere Application Server supports by configuration.

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. 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 configuration information and control for a cell. The deployment manager aggregates and communicates with the node agent processes on each node in the system.

- Node agent

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

- Application server

  Managed server that hosts J2EE applications.

Table 5 on page 57 shows the managed processes that are supported by each packaging option.

Table 5 WebSphere Application Server distributed administration support

	Express and Base	Network Deployment
Deployment manager	No	Yes
Node agent	No	Yes
Application servers	Stand-alone	Stand-alone or distributed server clustering

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

In a distributed server environment, the deployment manager maintains a master configuration repository that contains all of the cell's configuration data. The configuration repository at each node is a synchronized subset of the master repository. The node repositories are read-only for application server access. Only the deployment manager can initiate their update and push out configuration changes from the cell master configuration repository. It manages through communication with the node agent process resident on each node of the 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 of 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 or 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.

Developing and deploying J2EE applications

Figure 12 shows a high-level view of the stages of application development and deployment for J2EE applications.

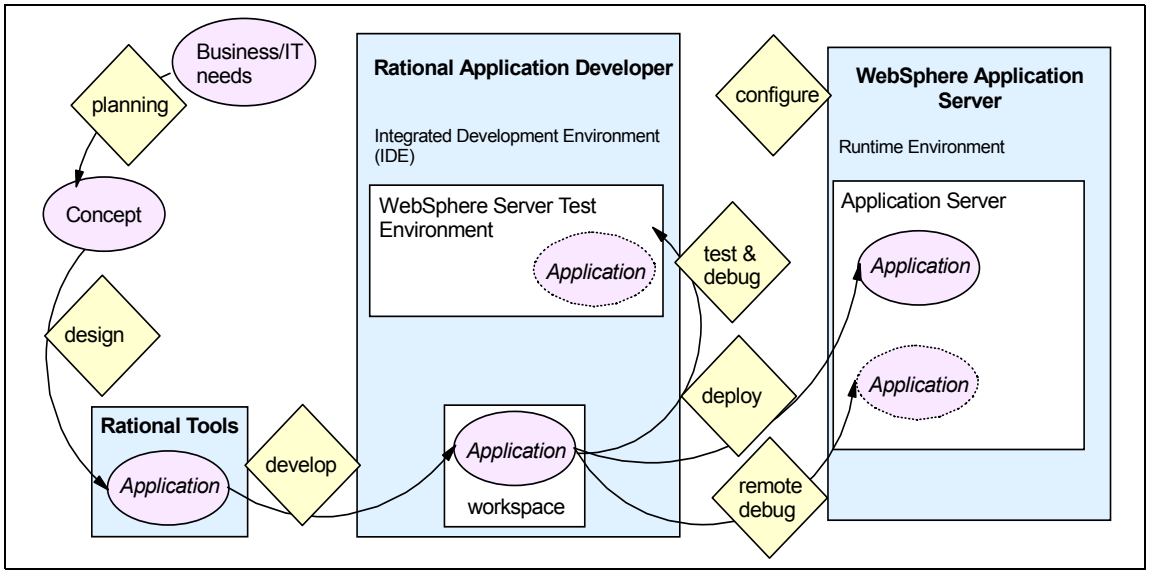

Figure 12 Develop and deploy

Application design

Design tools such as Rational Software Architect can be used to model the application using the Unified Modeling Language. The output of the modeling generally consists of use-case scenarios, class diagrams, and starter code that is generated based on the model.

Application development

Application development is performed using Rational Application Developer, the Application Server Toolkit, or a comparable IDE to create the enterprise application. You can start by importing pre-generated code from modeling tools, 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®, which enables multiple developers to share a single master source copy of the code.

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

Application packaging

J2EE applications are packaged into EAR files to be deployed to one or more application servers. A J2EE application contains any or all of the modules as shown in Table 6.

Table 6 J2EE 1.4 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 provides the Application Server Toolkit for packaging applications.

Enhanced EAR files

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

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

Application deployment

Applications are installed on application servers using the administrative console or the wsadmin scripting interface. You can deploy an application 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 have 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 message-driven beans.
- 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 has to be regenerated and copied to the Web server.

Application update

In previous releases, deploying an update to an application required deploying a complete EAR file and restarting the application. WebSphere Application Server V6 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, zipped files that contain application artifacts, or the complete application. All module types can be started (but only Web modules can be stopped).

Version 6 has a rollout start option for installing applications on a cluster that will stop, update, and start each cluster member in turn, ensuring availability.

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 into

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 has 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.

The flow of a J2EE application

Figure 13 shows the typical application flow for Web browser clients using either JDBC (from a servlet) or EJB to access application databases.

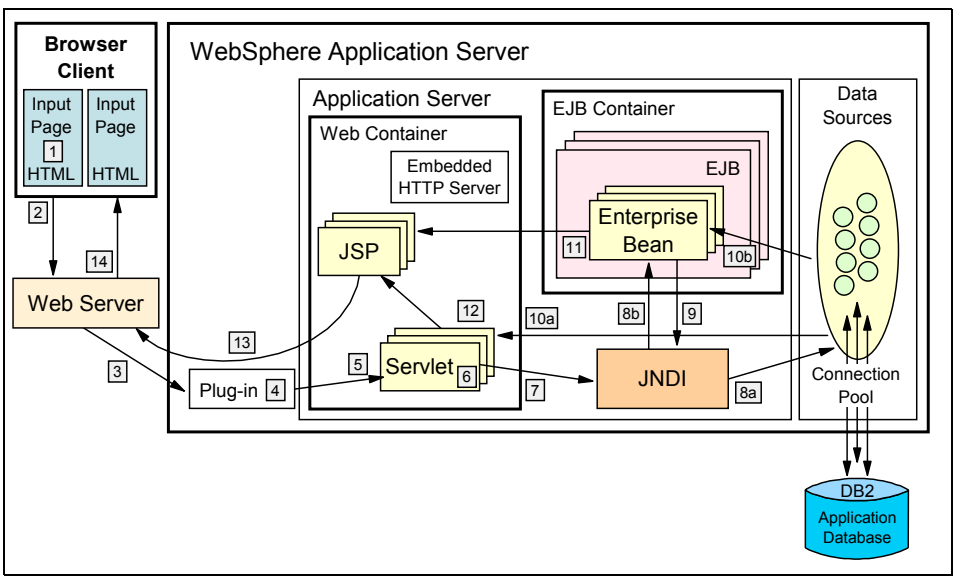

Figure 13 Application flow

The typical application flow is as follows:

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 Web server 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 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.
   - To the corresponding EJB container, which then instantiates the EJB when an EJB is requested.
9. If the EJB request involves an SQL transaction, it goes back to the JNDI to look up the datasource.
10. The SQL statement is executed and the retrieved data is sent back either to the servlet or to the EJB.
11. Data beans are created and handed off to JSPs in the case of EJBs.
12. The servlet sends data to JSPs.
13. The JSP generates the HTML that is sent back through the plug-in to the Web server.
14. The Web server sends the output page (output HTML) to the browser.

Technology support summary

Table 7 highlights the support that each WebSphere Application Server packaging option provides.

Table 7 WebSphere Application Server features and technology support

	Base and Express V6	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] ► Servlet 2.4 ► JSP 2.0 ► EJB 2.1 ► JMS 1.1 ► JTA 1.0.1B ► JavaMail 1.3 ► JAF 1.0.2 ► JAXP 1.2 ► JCA 1.5 ► Web Services 1.1 ► JAX-RPC 1.1 ► SAAJ 1.2 ► JAXR 1.0 ► JMX 1.2 ► JACC 1.0 ► JDBC 3.0, 2.1, Optional Package API (2.0) ► JNDI 1.2.1	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 messaging providers ► Message-driven beans	Yes	Yes
Web services runtime support	Yes	Yes

	Base and Express V6	Network Deployment V6
Security support ► Java 2 ► J2EE ► JACC 1.0 ► JAAS 1.0 ► CSIv2 and SAS authentication protocols ► LDAP or local operating system user registry ► LTPA authentication mechanism ► Kerberos, Technology Preview	Yes	Yes
► Simple WebSphere Authentication Mechanism	Yes	stand-alone server environment only
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	No	Yes
Caching Proxy	No	Yes
Dynamic caching	Yes	Yes
Performance and analysis tools		
Performance Monitoring Instrumentation (PMI)	Yes	Yes
Log Analyzer	Yes	Yes
Tivoli Performance Viewer (integrated into the administration console)	Yes	Yes
Administration and tools		
Administration and tools ► Web-based administration console ► Integrated IBM HTTP Server and Application Server Administration Console ► Administrative scripting ► Java Management Extension (JMX) 1.2 ► J2EE Management (JSR-077) ► J2EE Deployment (JSR-088) ► Application Server Toolkit	Yes	Yes

	Base and Express V6	Network Deployment V6
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
Programming Model Extensions[2]		
► Activity sessions ► Application Profiling ► Asynchronous Beans (now called WorkManager) ► Dynamic caching ► Dynamic query ► Internationalization Service ► Object Pools ► Scheduler Service (now called Timer Service) ► Startup Beans ► WorkArea Service ► Extended JTA Support ► Last Participant Support	Yes	Yes
► Back-up Cluster Support	No	Yes

1. You can see 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.

For more information

- Announcement

  http://www-306.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=an&subtype=ca&appname=GPA&htmlfid=897/ENUS206-076

- System requirements for WebSphere Application Server V6.1

  http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27007651

- WebSphere Application Server, Version 6.1 Information Center

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

- JSR168 Portlet Specification

  http://jcp.org/aboutJava/communityprocess/final/jsr168/

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

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

AIX®	Rational®	Tivoli®
ClearCase®	Redbooks (logo) ™	WebSphere®
IBM®	System i™	z/OS®
i5/OS®	System z™	

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaMail, JavaServer, JavaServer Pages, JDBC, JDK, JMX, JSP, JVM, J2EE, J2SE, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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