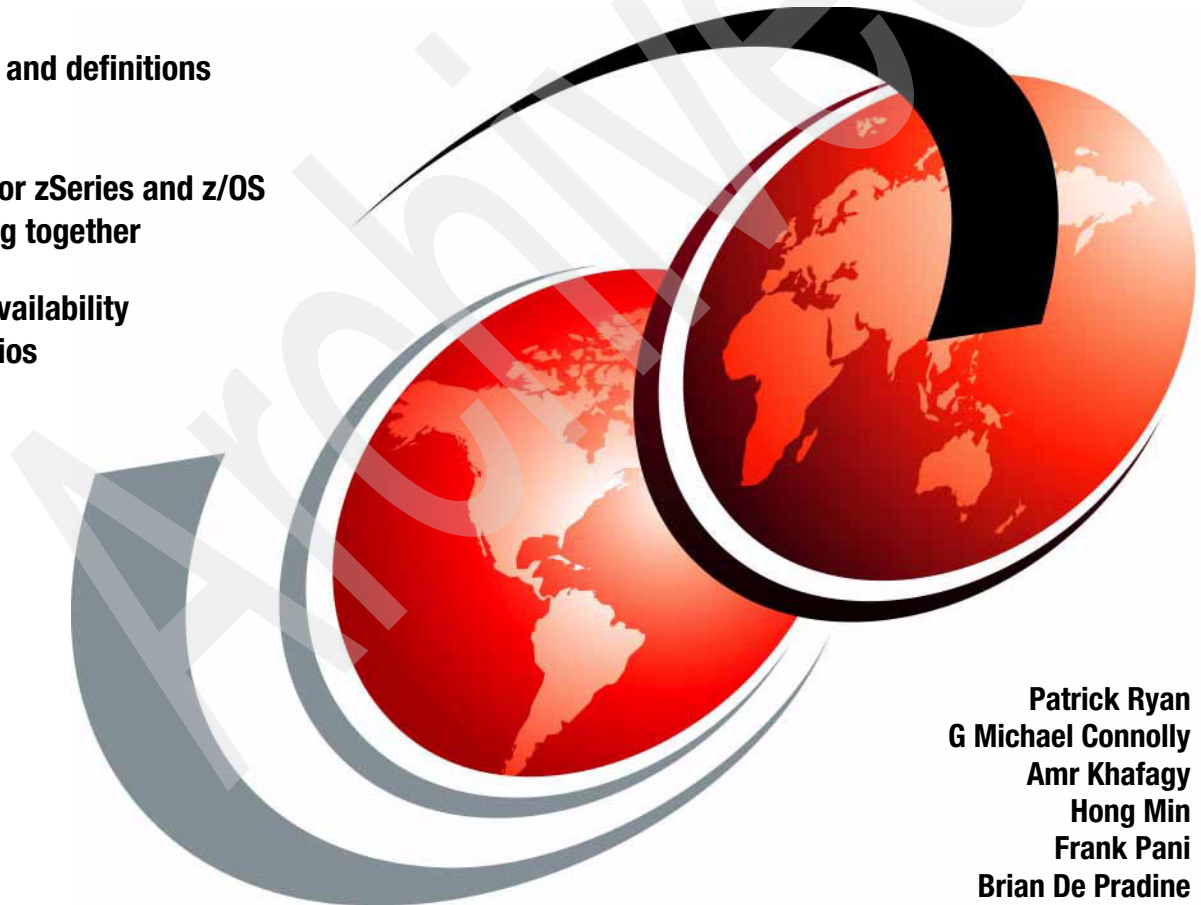


Architecting High Availability Using WebSphere V6 on z/OS

Theory and definitions

Linux for zSeries and z/OS working together

High-availability scenarios



Patrick Ryan
G Michael Connolly
Amr Khafagy
Hong Min
Frank Pani
Brian De Pradine



International Technical Support Organization

**Architecting High Availability Using WebSphere V6
on z/OS**

March 2006

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Third Edition (March 2006)

This edition applies to WebSphere Application Server for z/OS V6.0.1.

© Copyright International Business Machines Corporation 2002, 2004, 2006. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xi
Become a published author	xii
Comments welcome	xiii
Part 1. Theory	1
Chapter 1. Introduction	3
1.1 Availability	4
1.1.1 zSeries availability	4
1.1.2 z/OS availability	5
1.1.3 WebSphere Application Server for z/OS availability	5
1.1.4 Linux for zSeries availability	6
1.1.5 WebSphere Edge Components	6
1.1.6 Firewall availability	6
1.2 Scalability	7
1.2.1 zSeries scalability	7
1.2.2 z/OS scalability	7
1.2.3 WebSphere Application Server for z/OS scalability	8
1.2.4 Linux for zSeries scalability	10
1.3 J2EE application characteristics	10
1.4 Load balancing and affinity	14
1.4.1 Our environment	15
1.4.2 Load balancing with session persistence	16
1.4.3 Load balancing with HTTP session affinity	16
Chapter 2. Considerations for scalability	19
2.1 Definitions	20
2.1.1 Performance	20
2.1.2 Scalability	20
2.1.3 Throughput	20
2.1.4 Load balancing and workload management	21
2.1.5 Availability	21
2.1.6 Recovery	24
2.1.7 Maintainability	25
2.1.8 Session management	26

2.1.9 Security	27
2.2 Increasing performance.	27
2.2.1 Performance definition	27
2.2.2 Performance measurement.	28
2.2.3 Tuning performance	29
2.3 Scaling your environment	33
2.3.1 WebSphere scaling anatomy	34
2.3.2 Scaling process.	35
2.3.3 Understanding your workload	38
2.3.4 Scaling techniques	41
2.4 Increasing throughput	44
2.4.1 Caching.	45
2.5 Application programming patterns.	47
2.5.1 Applicable patterns	47
2.5.2 Other considerations.	48
Chapter 3. Workload balancing	53
3.1 Workload balancing.	54
3.2 Domain Name Server mapping solutions	55
3.2.1 Round-robin DNS	55
3.2.2 Connection optimization (DNS/WLM)	57
3.3 Connection dispatching solutions	58
3.3.1 Load Balancer	58
3.3.2 Sysplex Distributor	62
3.3.3 Sysplex Distributor and MultiNode Load Balancing (MNLB)	66
Chapter 4. HTTP sessions	69
4.1 Overview	70
4.2 Maintaining session state	70
4.2.1 Client tier	71
4.2.2 Web tier - HTTP sessions	72
4.2.3 Session affinity	74
4.2.4 Session recovery and failover.	76
4.2.5 Database session persistence	77
4.2.6 Memory-to-memory session replication	78
4.3 Data flow in HTTP sessions with affinity	84
4.3.1 Session persistence using an internal replication domain	85
4.3.2 HTTP session failover tests	90
4.4 HTTP sessions versus stateful session beans	94
Chapter 5. Architectures for availability	99
5.1 Things to consider.	100
5.2 Logical tiers	101
5.3 Logical architectures	102

5.3.1	Two-tier logical architecture	102
5.3.2	Three-tier logical architecture	103
5.4	Physical architectures	104
5.5	z/OS base infrastructure	105
5.6	Two-tier physical architecture	106
5.7	Three-tier physical architectures	112
5.7.1	Decouple Web/EJB and EIS layers	113
5.7.2	Decouple Web and EJB layers	118
Chapter 6.	WebSphere platform messaging	123
6.1	Overview	124
6.2	Concepts and architecture	124
6.2.1	Buses	124
6.2.2	Bus members	124
6.2.3	Messaging engines	125
6.2.4	Data stores	127
6.2.5	Destinations	128
6.2.6	Mediations	129
6.2.7	Foreign buses	130
6.3	Availability	130
6.3.1	Failover	130
6.3.2	Messaging engine clustering	133
6.4	Scalability	134
6.5	Workload balancing	135
6.5.1	Connecting to a service integration bus	136
6.5.2	Push workload balancing	140
6.5.3	Pull workload balancing	142
6.5.4	WLM Classifier	144
6.6	Core group policies	145
6.7	Things to consider when writing applications	146
Part 2.	System setup	147
Chapter 7.	Setting up the infrastructure	149
7.1	Infrastructure considerations	150
7.2	The architecture we used	150
7.2.1	The front-end systems	151
7.2.2	The back-end systems	153
7.3	Setting up the Network Dispatcher	156
7.3.1	Linux for zSeries installation	156
7.3.2	Configuration methods	159
7.3.3	Load Balancer configuration	161
7.3.4	High availability configuration	161
7.3.5	Web server forwarding configuration	164

7.3.6 z/Series TCP configuration (Layer 2 vrs Layer 3)	165
7.4 Setting up the IBM HTTP Server on zLinux	167
7.4.1 IBM HTTP Server configuration	168
7.4.2 WebSphere Edge Components/Web server plug-in	168
7.4.3 Configuring the WebSphere plug-in	169
7.4.4 How the front-end traffic flows	174
Chapter 8. Setting up the z/OS infrastructure	177
8.1 Sysplex Distributor	178
8.1.1 The implementation we used	179
8.2 Setting up WebSphere	185
8.2.1 Target infrastructure	186
8.2.2 HFS setup for the SMP/E environment	186
8.2.3 Setup of the WebSphere libraries	187
8.2.4 WebSphere configuration	188
8.2.5 IP name on the Location Service daemon definition	195
8.2.6 Port definitions and assignments	196
8.2.7 Define the first server	200
8.2.8 Modifying the port settings	204
8.2.9 New server RACF definitions	206
8.2.10 Virtual host alias definitions	207
8.2.11 Defining the cluster	208
8.2.12 Web container definitions	213
8.2.13 How these definitions fit together	217
8.3 Other options to set up	219
8.3.1 Setup for memory-to-memory internal replication	219
8.3.2 Setup for session persistence with DB2	225
8.3.3 Peer Restart and Recovery function	228
8.3.4 Sysplex Failure Management (SFM)	230
8.4 Maintenance strategy	231
8.4.1 The HFS structure for upgrades	232
8.4.2 Applying maintenance	233
8.4.3 HFS versus zFS	234
Chapter 9. Setting up the messaging infrastructure	237
9.1 Set up the data stores	238
9.1.1 Create the data store tables	238
9.1.2 Create the JDBC data source for the messaging engines	241
9.2 Configure the service integration bus (SIB)	243
9.2.1 Create the bus	243
9.2.2 Add the cluster to the bus	243
9.2.3 Add additional messaging engines	245
9.3 Configure preferred servers	246

9.3.1 Set up core group policies	247
9.4 Verify the configuration	251
Part 3. Availability tests	255
Chapter 10. Applications used to drive the workload	257
10.1 Applications used to create the workload	258
10.1.1 TraderDB2	258
10.1.2 TraderJMS	259
10.2 WebSphere Workload Simulator	260
10.2.1 The workload	262
10.2.2 Scripts for WebSphere Studio Workload Simulator	263
10.3 Setting up TraderJMS	274
10.4 Configuring the required resources	274
10.4.1 Configuring service integration bus (SIB) resources	274
10.4.2 Configure default messaging provider resources	275
10.4.3 Configure JDBC resources	278
10.5 Install the TraderJMS application	279
Chapter 11. Planned infrastructure outages	285
11.1 Installing new maintenance levels of WebSphere for z/OS	286
11.1.1 Our installation's example to roll the upgrade	286
11.1.2 Implementing the upgrade on system SC54	287
11.1.3 Rolling Upgrade process	289
Chapter 12. Unplanned outages	297
12.1 Summary of unplanned outage tests	298
12.2 Edge server	299
12.3 Proxy server	300
12.4 HTTP server	301
12.5 Single WebSphere	301
12.6 WebSphere servant process	302
12.7 Two WebSpheres	304
12.8 Sysplex Distributor	306
12.9 Deployment Manager with Peer Recovery	307
12.10 WebSphere control region adjunct	310
12.11 Summary of test results	313
12.12 End-to-end monitoring	313
Chapter 13. Application availability tests	317
13.1 New Application Update features	318
13.1.1 Finer grain application update	318
13.1.2 Rollout Update	318
13.2 Our test	319

13.2.1 Rollout Update of the entire TraderDB ear file	319
13.2.2 Single module update	323
13.2.3 Single file update.....	328
Related publications	333
IBM Redbooks	333
Other publications	335
Online resources	336
How to get IBM Redbooks	336
Help from IBM	336
Index	337

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.

Trademarks

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

AIX®	eServer™	Redbooks™
CICS®	Geographically Dispersed	Redbooks (logo)  ™
CICSplex®	Parallel Sysplex™	RMF™
Distributed Relational Database	GDPS®	Tivoli®
Architecture™	IBM®	Virtualization Engine™
DB2 Connect™	IMS™	VTAM®
DB2 Universal Database™	MVS™	WebSphere®
DB2®	OS/390®	z/OS®
DRDA®	Parallel Sysplex®	z/VM®
@server®	RACF®	zSeries®

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

In this IBM® Redbook, we describe how to configure the various components of an e-business solution to exploit the availability and scalability benefits of zSeries® and Parallel Sysplex® using multiple LPARs running Linux® for zSeries and z/OS®.

This publication applies to WebSphere® for z/OS V6, and is a continuation of the project evaluating high availability using WebSphere for z/OS V5 and V4. Presented in three parts (theory, systems setup, and availability tests), we cover workload balancing, the use of HTTP sessions, various architectures, and how to set up and test your infrastructures.

Considerations for configuring the systems and applications in an e-business environment are also examined, and we address such questions as:

- ▶ Should there be separate front-end systems for running the WebSphere Application Server?
- ▶ How should e-business components be configured to minimize or eliminate the impact of a system outage?
- ▶ What is the best way to communicate between the WebSphere systems and the back-end application systems?
- ▶ What is the impact on the end user if there is a failure in one of the front-end or back-end systems?

The team that wrote this redbook

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

Patrick Ryan is a Senior Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of the WebSphere Application Server for z/OS. He has more than 30 years of experience with mainframe operating systems, and more than five years of experience with WebSphere Application Server for z/OS.

Amr Khafagy is a Certified IT Specialist, working for IBM Canada. He has 20 years of experience in systems programming. His areas of expertise include z/OS, WebSphere, UNIX® System Services and Storage. He is also an IBM

Certified Advanced System Administrator - WebSphere Application Server for z/OS V5.1, and co-authored several IBM Redbooks™ about z/OS and WebSphere on z/OS. He holds a Bachelor's degree in Engineering.

Brian DeP is a developer working for IBM Software Group in Hursley, England. He has worked for IBM for seven years, developing various products on mainframe systems. For the past three years he has worked on WebSphere Application Server development. He has a Bachelor of Science in Applied Physics from Columbia University, and is currently working towards a Master of Science in Software Engineering from Oxford University.

Frank Pani is a Technical Services Professional for Host Software Services, IBM Global Services, Canada. Since June 2000 he has worked on developing DB2® solutions in Java™ for OS/390®. He holds a degree in Combinatorics & Optimization from the University of Waterloo, ON, Canada.

Hong Min is an IT specialist at the IBM Design Center for business on demand, Poughkeepsie, USA. She has eight years of experience helping customers enable their e-business configurations on z/OS using IBM technologies. Her technical interests include WebSphere, J2EE™ applications, Grid computing and zSeries.

G Michael Connolly is an IT consultant at the International Technical Support Organization, Poughkeepsie Center. He has more than 30 years of IBM software development experience in both distributed systems and the mainframe zSeries. He holds a Bachelor of Arts in Humanities from Villanova University. His areas of expertise include TCP/IP Communications, UNIX System Services, WebSphere for z/OS, and most recently the IBM Virtualization Engine™ EWLM product.

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

Rich Conway, Gregory Geiselhart, Bill White, Roy Costa, and David Bennin of the International Technical Support Organization, Poughkeepsie Center.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Archived



Part 1

Theory

In this part we introduce the concepts involved in creating a highly available WebSphere architecture on zSeries, and explore various alternatives for this architecture.

Archived

Introduction

To build a high-availability e-business infrastructure, the hardware, software and network must be chosen and configured to minimize or avoid any disruption or failure so that clients perceive a continuously available system. When you use the Patterns for e-business methodology (see the Patterns-related redbooks on www.redbooks.ibm.com), one aspect of system design is designing for availability and scalability.

In this chapter we describe availability, scalability, and reliability—and how the zSeries solution differs from other platforms and operating systems. We also introduce the WebSphere Edge Components and WebSphere Application Server for z/OS.

We discuss the following topics:

- ▶ Availability
- ▶ Scalability
- ▶ J2EE Application characteristics
- ▶ Load balancing and affinities

1.1 Availability

In this redbook, we discuss the setup of a high-availability e-business infrastructure using WebSphere Application Server for z/OS front-ended by HTTP servers on Linux for zSeries.

On our Linux for Series front end, we configured a load balancer to distribute TCP/IP traffic in a round robin fashion to the two HTTP servers, which were also built on the front end LPAR. These two HTTP servers were enabled with a plugin file that could honor session affinity to the back-end z/OS server regions.

The zSeries platform differs from other server platforms in terms of hardware and software reliability. That is why we chose to use the zSeries platform for our high-availability environment. Let's talk about some of these differences.

1.1.1 zSeries availability

The zSeries server architecture includes self-healing capabilities to prevent downtime caused by system crashes.

The latest zSeries RAS strategy is a building-block approach developed to meet customers' stringent requirements of achieving Continuous Reliable Operation (CRO). The building blocks are:

- Error Prevention
- Error Detection
- Recovery
- Problem Determination
- Service Structure
- Change Management
- Measurement and Analysis

The initial focus is on preventing failures from occurring in the first place. This is usually accomplished by using highest reliability components from our technology suppliers, and using screening, sorting, burn-in, run-in, and by taking advantage of technology integration. For Licensed Internal Code (LIC) and hardware design, failures are eliminated through rigorous design rules, design walkthroughs, peer reviews, element/subsystem/system simulation, and extensive engineering and manufacturing testing.

The z990 RAS strategy is focused on recovery design necessary to mask errors and make them "transparent" to customer operations. Extensive hardware recovery is implemented to be able to detect and correct array faults. In cases where total transparency cannot be achieved, the capability exists for you to restart the server with the maximum possible capacity.

For more information on the latest zSeries hardware availability and scalability items, see *IBM eServer zSeries 990 Technical Guide*, SG24-6947.

1.1.2 z/OS availability

The z/OS operating system has roots that go back to the early days of MVS™, which was designed over 30 years ago with high availability in mind. The operating system takes advantage of the self-healing attributes of the hardware, and extends them by adding functions such as recovery services for all operating system code, address space isolation, and storage key protection. Functions such as Workload Manager (WLM), Resource Recovery Services (RRS), and Automatic Restart Manager (ARM) assure the availability of applications.

The z/OS operating system can be configured into a Parallel Sysplex, which is the clustering technology for the mainframes. The main objective of a Parallel Sysplex is continuous availability without compromising perceived client performance.

High availability requires at least two servers providing the same service to their clients, so that these servers allow for the recovery of service when failures occur. That is, servers perform some sort of backup function for each other within the cluster. High availability minimizes unplanned outages.

Continuous availability is achieved with the zSeries and z/OS Parallel Sysplex clustering technology. This technology implements a data-sharing design that allows a database to be concurrently read and updated by application clones running on multiple z/OS images on one or more physical servers. Continuous availability avoids or minimizes unplanned outages (high availability) and reduces or eliminates planned outages.

The Workload Manager balances application workload across the systems in the Parallel Sysplex. If there is a failure or a planned outage on one system, other systems within the Parallel Sysplex take over the full workload. By implementing Geographically Dispersed Parallel Sysplex™ (GDPS®), the servers can be as far as 40 km apart from each other, thus avoiding a total site-wide disaster.

Using the Parallel Sysplex clustering architecture, you can achieve a near continuous availability of 99.999%, or 5 minutes of downtime a year.

1.1.3 WebSphere Application Server for z/OS availability

The WebSphere Application Server for z/OS takes full advantage of the high availability features provided by z/OS Parallel Sysplex and zSeries. WebSphere for z/OS incorporates some unique code designed to take advantage of z/OS strengths. It has recovery termination management that detects, isolates,

corrects and recovers from software errors; the ability to differentiate and prioritize work based on service level agreements and clustering capabilities; and the ability to make nondisruptive changes to software components, such as z/OS, WebSphere Application Server, and resource managers. It also exploits 2-phase commits to handle transactions in a heterogeneous environment. If a server fails, other servers in the cluster take over the work and the server is recovered.

High availability manager

WebSphere Application Server V6 includes a new high availability manager component whose main function is to eliminate single points of failure. This component runs inside every JVM™, that is, within a WebSphere Application Server cell. The High Availability manager provides peer-to-peer failover for critical services by always maintaining a backup for these services.

1.1.4 Linux for zSeries availability

Most Linux for zSeries environments are run on the z/VM® operating system, although they could run directly on the zSeries processor or LPAR.

Linux for zSeries inherits the zSeries hardware's reliability, but software faults can still cause outages, both from the z/VM operating system or from the Linux for zSeries operating system. Configuring a highly available e-business infrastructure must take advantage of the z/VM availability items such as hardware error recovery, automation of starting and stopping Linux guests, heartbeat mechanisms between two z/VM systems for takeover functions, clustering, and DASD sharing. See *Linux on IBM zSeries and S/390: High Availability for z/VM and Linux*, REDP-0220 for more detail on z/VM and Linux availability items.

1.1.5 WebSphere Edge Components

High availability is achieved through parallelism, load balancing, and failover support. In our environment, we ran the backup WebSphere Edge Components on a separate z/VM, but they were on the same physical zSeries machine. To achieve total availability, the WebSphere Edge Components should have been placed on separate physical machines.

1.1.6 Firewall availability

For the purposes of this redbook we did not include firewalls as part of our configuration. As a normal part of a high availability configuration, all firewalls should be configured with backups and failover mechanisms.

1.2 Scalability

1.2.1 zSeries scalability

The IBM @server® zSeries z990 is the latest addition to the zSeries server family. Each z990 server can have up to 32 central processors and support up to 30 logical partitions (LPARs). Each LPAR can run different operating systems, while being managed by a central hardware console.

1.2.2 z/OS scalability

Parallel Sysplex architecture is designed to integrate up to 32 systems in one cluster. Each of these systems can handle multiple different workloads and access databases using data sharing technology. With zSeries, a new function called Intelligent Resource Director (IRD) was introduced, which reassigns resources on demand to different logical partitions.

Parallel Sysplex architecture provides performance for workloads with unpredictable demands: It can run WebSphere and traditional OLTP/DB applications simultaneously at 100% utilization, as shown in Figure 1-1. More work is processed in a single server, without over-configuring for complementary peaks.

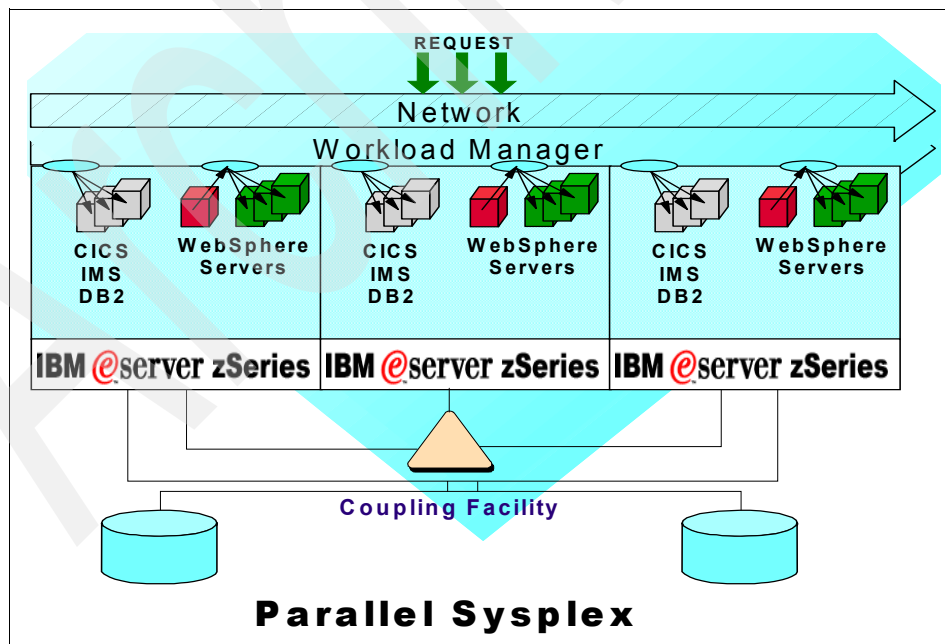


Figure 1-1 Anatomy of an enterprise Parallel Sysplex

1.2.3 WebSphere Application Server for z/OS scalability

Figure 1-2 shows the basic WebSphere Application Server for z/OS architecture. We will describe the components in this architecture and later expand upon it to show how it works within a Parallel Sysplex as shown in 2.3.1, “WebSphere scaling anatomy” on page 34.

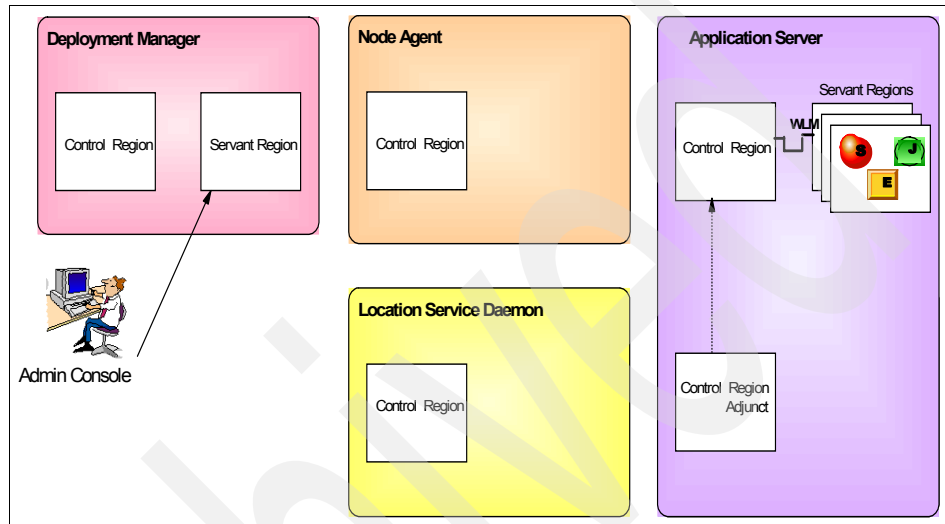


Figure 1-2 Basic WebSphere Application Server for z/OS architecture

The basic WebSphere applications server on z/OS consists of the following parts:

- ▶ Deployment Manager
- ▶ Node Agent
- ▶ Location Service Daemon
- ▶ Application Server

Deployment Manager

The Deployment Manager provides an interface that allows you to configure and deploy applications to the server. It consists of a Control Region (CR) and a single Servant Region (SR) to which the administrative console is attached.

Node Agent

The Node Agent represents the physical environment (or Node) where the application server or servers are located. This consists of a single CR.

There can be more than one application server defined on a given node, but there will only be a single Node Agent that represents all of them. The Node Agent represents each physical system, not each logical system, within a cell.

The node shares a common repository and has a common namespace. There should be a clear policy in an installation as to which entities (server, node, or cell) should be the separation level for production, test, and development environments for multiple applications.

Location Service Daemon

The Location Service Daemon provides a lookup service for the application servers. Like the Node Agent, it also consists of a single CR, and there is only one Daemon for each node.

Application Server

The Application Server consists of a CR and one or more SRs. The Controller or CR controls and routes work into the SRs. It dynamically starts and stops SRs based on the needs of the workload, based on input from the z/OS WorkLoad Manager (WLM).

The SR contains two containers, namely the Web Container and the J2EE Container. The SR is a separate z/OS address space and can run in parallel with other SRs in a z/OS LPAR.

There can be multiple application servers defined on a given node, each containing a different mix of applications.

CRA

The Application Server may also include a Control Region Adjunct (CRA) that contains the messaging resources for the new Service Integration Bus (SIB), available in WebSphere Application Server V6 for z/OS. This includes support for the underlying protocol for JMS connections to the new default messaging provider, which is based on SIB technology.

The CRA will only be present if the SIB service is explicitly activated.

The Control or CR controls and routes work into the server regions. It dynamically starts and stops SRs based on the needs of the workload, consulting the Workload Manager. If an error occurs in one SR, the CR can start an SR dynamically, thus supporting availability. Multiple protocols are supported, HTTP(S) and IIOP.

For more information about asynchronous messaging see Chapter 6., “WebSphere platform messaging” on page 123.

1.2.4 Linux for zSeries scalability

The flexibility of Linux combined with zSeries virtualization allows you to get systems up and running in minutes and make them scalable. Since Linux for zSeries takes advantage of the traditional zSeries hardware qualities of service and lives close to the WebSphere Application Server residing on z/OS, it makes the Linux for zSeries an ideal environment to create a DMZ consisting of WebSphere Edge Components, firewalls, and HTTP servers.

Inside the VM LPAR, you can have many Linux images running at the same time. These Linux images can be built into network dispatchers (which have the ability to load-balance TCP/IP traffic to back-end hosts), HTTP servers (which have the ability to hold the plugin to the back-end z/OS WebSphere Application Server system), and firewalls (which have the ability to protect the front DMZ side from attacks and intruders from the Internet). It's very easy to build and manage any of these servers on the fly. The ease of management comes from the advantage of being "virtualized" and that you can add resources such as disk or CPU whenever you want.

1.3 J2EE application characteristics

In this redbook, we discuss the setup of a high availability e-business infrastructure using WebSphere Application Server for z/OS front-ended by a Linux for zSeries HTTP server, with the WebSphere plug-in. We are also using the WebSphere Edge Components for workload balancing.

WebSphere Application Server for z/OS is a J2EE-compliant application server. In "J2EE standard" on page 10 we give a brief overview of the J2EE specification and J2EE applications. We also explain why and how a J2EE infrastructure has to deal with HTTP sessions.

J2EE standard

The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications.

You can find a detailed description of the J2EE standard at:

<http://java.sun.com/j2ee/overview.html>

WebSphere Application Server V6.01 offers production-ready J2EE 1.4 standards and Web services support.

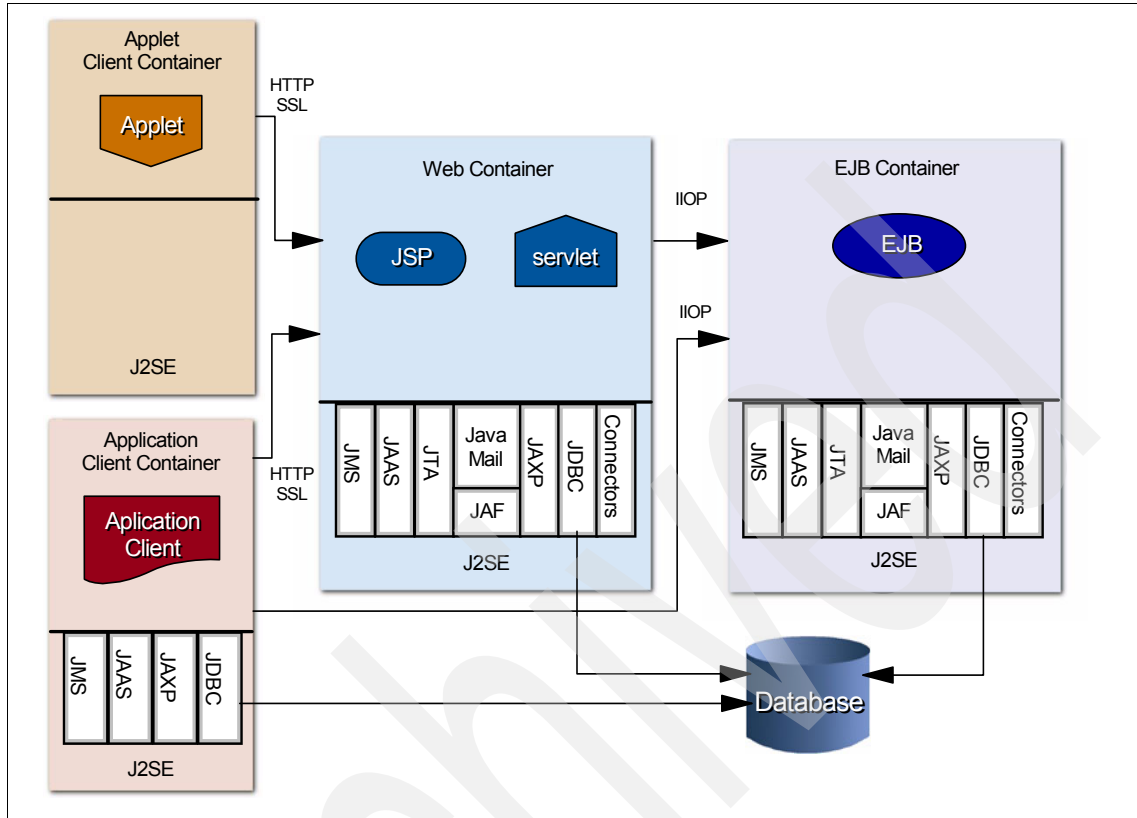


Figure 1-3 J2EE architecture

J2EE applications

The J2EE programming model has four types of application components, as shown in Figure 1-3:

- ▶ Application clients
- ▶ Applets
- ▶ Servlets and JavaServer™ Pages™
- ▶ Enterprise JavaBeans™

Application clients

Application clients are Java programs that typically run on a desktop computer with a graphical user interface (GUI). They have access to the full range of J2EE server-side components and services.

The application client component is often used where the user interface capabilities of a standard Web browser are not considered sufficient.

Applets

An applet is a client Java class that typically executes in a Web browser, but can also run in a variety of other client applications (for instance, Java Web Start) or devices.

Applets are often used in combination with HTML pages to enhance the user experience provided by a Web browser. They can also be used to shift some of the processing workload from the server to the client.

Servlets and JavaServer Pages

Servlets and JavaServer Pages (JSPs) are server-side components used to process requests from HTTP clients, such as Web browsers. They handle presentation and control of the user's interaction with the underlying application data and business logic. They can also generate formatted data, such as XML, for use by other application components.

Servlets and JavaServer Pages are deployed, managed, and executed in a J2EE Web container and are often called "Web components".

Enterprise JavaBeans

The Enterprise JavaBeans (EJB™) specification is a foundation for the Java 2 Platform, Enterprise Edition (J2EE) defined by Sun™ Microsystems. Vendors use this specification to implement an infrastructure in which components can be deployed, and use a set of services such as distributed transactions, security, or life-cycle management. As a developer, you just reuse the services detailed in the specification. For example, you do not need to include any code in your components to make them transactional. This lets you concentrate on the business logic of the application.

Enterprise beans are designed to be portable from one vendor's execution environment to another, independent of the choices made by the vendor to implement the services described in the specification.

The quality of service required by an enterprise bean is described outside of the component in a *deployment descriptor*. The deployment descriptor is analyzed at deployment time by a tool provided by the vendor. This feature provides a great deal of flexibility for reusing your component. For example, if you wish to change the transactional behavior of an enterprise bean, you need to change only the transaction attribute stored in the deployment descriptor, not the EJB business logic. Changes are taken into account when you redeploy the enterprise bean in the container.

There are three types of enterprise beans: *session beans*, *entity beans*, and *message-driven beans*. Session beans and entity beans provide interfaces to allow clients access to their business methods. Message-driven beans, however,

do not provide any interfaces. Instead, they are used to asynchronously consume messages from a destination.

Session beans

A session bean is used to capture the business logic associated with various tasks. This logic can even be used to coordinate the actions of other enterprise beans in an application. There are two types of session bean: *stateless* and *stateful*. A stateless session bean does not maintain any state between method invocations. A stateful session bean, however, can maintain state from one method invocation to the next. This information is lost when the bean instance is destroyed.

Entity beans

Entity beans represent persistent data, typically a row in a database table. In this case, creating an entity bean is equivalent to adding a record to the table. Even if the bean instance is removed, or if the server crashes, the bean instance can be recreated from its persistent state.

Message-driven beans

Message-driven beans have been available since J2EE 1.3. The only way that a client can interact with a message-driven bean is to send a message to the destination to which it is listening. Message-driven beans can be used in the same role as stateless session beans, that is, to capture the business logic associated with an application.

For more details about J2EE and J2EE applications, see *Rational Application Developer V6 Programming Guide*, SG24-6449.

Web client

In addition to the application client, there is a particular type of client for a J2EE application called *Web client*. Web clients constitute the majority of users of WebSphere today. (The configuration in this redbook, as well as much of the discussion about the highly available configurations in this book, is specific to Web clients.)

Such a client uses a Web browser to interact with WebSphere Application Servers through the intermediary of a Web server, which in turn invokes a servlet within WebSphere. The transport protocol used between a Web client and a Web server is HTTP, a stateless protocol.

1.4 Load balancing and affinity

Establishing a highly available and scalable infrastructure means installing redundant components. Load balancing is the mechanism for distributing the incoming work between the redundant components. Load balancing improves the availability and scalability by transparently clustering servers. Scalability is greatly improved because the servers can be added or removed transparently, as needed.

An e-business infrastructure usually consists of many different tiers (Edge servers, Web and Web Application servers, directory and security services, data and transaction servers, and storage systems). Therefore, it is necessary to employ load balancing techniques at each tier of the infrastructure for a highly available infrastructure. For example, there must be load balancing between the Web servers, then the application servers, and from there to the backend systems.

There are different methods of doing the load balancing; refer to Chapter 3., “Workload balancing” on page 53 for more information. Each method uses a different approach to decide how to distribute work among the components it serves.

WebSphere Edge Components can be used in conjunction with the WebSphere Application Server to control client access to Web servers and to enable business enterprises to provide better service to users who access Web-based content over the Internet or a corporate intranet.

Using Edge components can reduce Web server congestion, increase content availability, and improve Web server performance. As the name indicates, Edge components usually run on machines that are close (in a network configuration sense) to the boundary between an enterprise’s intranet and the Internet, typically deployed in a DMZ setting.

For further information, see Part 2. “Distributing the workload” in the redbook *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392.

1.4.1 Our environment

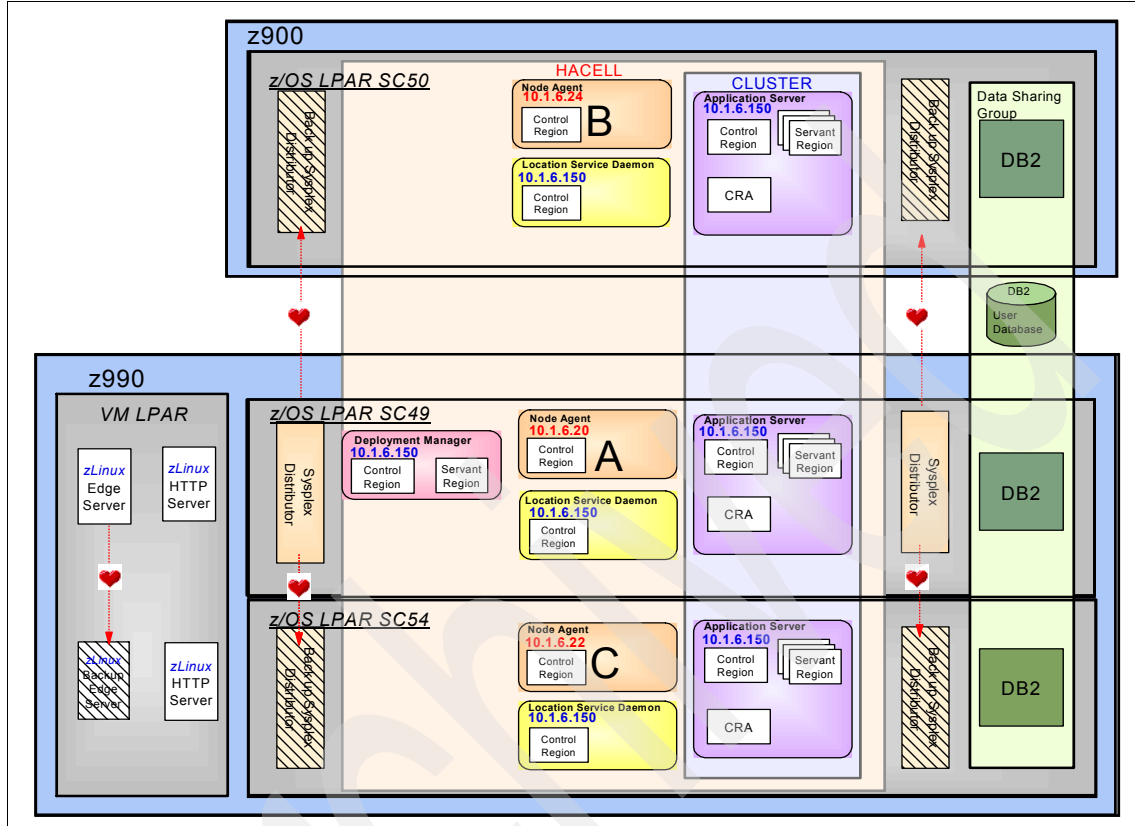


Figure 1-4 Our high availability test environment

As shown in Figure 1-4, we created a redundant, scalable, reliable e-business infrastructure. We will discuss the setup of this environment further in Chapter 7, “Setting up the infrastructure” on page 149.

For the utmost freedom in workload balancing, it is necessary that there be no affinity to one of the components. To achieve this goal from a functional perspective, you must ensure that each system is a clone of the other. By “clone” we mean that there are no specific functions assigned to just one system—instead, all systems must have the same base and application code so that they can process all application requests.

However, there could be reasons why you would want certain application requests to run on a certain component; an example of this is HTTP session

affinity. For more information about the use of HTTP sessions, see 4.2.3, “Session affinity” on page 74.

If your application uses HTTP sessions, you must decide how to ensure that each individual request belonging to an HTTP session is able to get its session information. You can do this by storing your HTTP session information in a shared DB2 database where all application servers can access it (this is known as *session persistence*), or you can keep the session data in the memory of one application server and make sure that each request in the session is sent to this same server (this is known as *session affinity*). Within a server cluster you may also use Data Replication Services to replicate session information.

1.4.2 Load balancing with session persistence

Using this solution offers several advantages. Storing session data in a shared database means it doesn't matter to which system and to which clone of your application a request is routed—the session information can be found in the database. The workload balancing methods do not have to choose a specific system.

You also have the advantage that if one system or one application server instance fails, the HTTP session can be taken over by a surviving server instance; only the active request fails, not the whole session.

The disadvantage of this solution is the overhead of storing and restoring the session data each time a new request is made. This can lead to a performance impact, depending on how big your session object is.

Another issue you must be aware of is that in order to store the session object to a database, your HTTP session objects must be serialized.

1.4.3 Load balancing with HTTP session affinity

If it is acceptable to your application that a user loses their active session in the case of an application server outage, then you do not need to store your session information in a shared database. You can use HTTP session affinity instead to find the right session data for your request. In this case, the session information is kept in the memory of the application server that established the session.

The workload balancing method you choose has to ensure that a request belonging to a specific session is routed to the application server environment where its session data is held.

There is no stored session information in the database. Therefore, you avoid the overhead of accessing the database and will get better performance for the application.

The drawback of a session affinity solution is that if the server instance fails, the session is lost—meaning that not only is the current request lost, but all subsequent requests on this session will also fail. WebSphere V5 improved this limitation by introducing the Data Replication Service. DRS copies the session information from one server instance to the other, so in case of a failure, the transaction is not lost.

For more information about handling session affinity in a Parallel Sysplex, see 4.2.3, “Session affinity” on page 74 and 4.2.6, “Memory-to-memory session replication” on page 78.

Archived

Archived

Considerations for scalability

The objective of this chapter is to explain how a basic WebSphere configuration can be enhanced and extended to provide more power by better exploiting the available resources of each system, and in addition by using multiple systems.

We define system configurations that have the following properties:

- ▶ Performance
- ▶ Scalability
- ▶ Throughput
- ▶ Load balancing and Workload Management
- ▶ Availability
- ▶ Maintainability
- ▶ Session management
- ▶ Security

Important: In addition to defining the system configurations that support and exhibit the above properties, we must develop applications that can take advantage of these and not introduce limitations into the infrastructure that inhibit any of the above factors.

2.1 Definitions

With a badly performing application, scaling the application server environment may not help. The assumption for scaling an infrastructure is that the application is itself inherently scalable. Application considerations are discussed further in 2.5, “Application programming patterns” on page 47.

The following provides definitions for the properties that a highly available infrastructure should exhibit.

2.1.1 Performance

Performance involves minimizing the response time for a given load while at the same time minimizing the system resource consumption for each transaction.

2.1.2 Scalability

Any configuration that is scalable should allow the overall system to service a higher client load than that provided by the simple basic configuration. Ideally, it should be possible to service any given load simply by adding the appropriate number of servers or systems.

Scale can be achieved in two ways, vertical and horizontal.

Vertical scaling

Vertical scaling involves creating application server clones or additional cluster members on the same physical system, or node. This allows more throughput to be achieved on one physical system and may allow the system’s processing power to be more efficiently allocated.

Horizontal scaling

In horizontal scaling, cluster members are created on multiple physical systems/nodes. This allows a single WebSphere application to run on several systems while still presenting a single application image, making the most effective use of the resources of a distributed computing environment.

2.1.3 Throughput

While related to performance, throughput more precisely defines the number of concurrent transactions that can be accommodated. This is often measured in terms of kilobytes per second that can be delivered through the front (Web tier) of the system.

This figure can, however, be distorted if the page weights involved in the application contain large volumes of data. This may drive the throughput up to a high level for very few transactions.

Page weight is defined as the amount of information contained in one complete page, measured in kilobytes. This includes all text, script, and graphics that is sent to the user's browser.

2.1.4 Load balancing and workload management

Load balancing and workload management are the means by which we ensure that each system or server in the configured infrastructure processes a fair share of the overall load that is being processed by the whole infrastructure. In other words, one system should never be overloaded while another system is mostly idle.

If all systems are of approximately the same processing power, each should process a roughly equal share of the load. If the various systems in the infrastructure are of different processing capabilities (for example, a PC vs. a large AIX® system), each should process a portion of the load that is proportional to its relative processing power.

In addition, workload management within z/OS manages the available system resources more directly and will make additional resources available to better service higher priority workloads.

If the total load being served by the infrastructure changes over time, the systems involved should naturally adapt themselves to maintain an even distribution of the total load. On z/OS, this is achieved by the WorkLoad Manager subsystem.

Whereas in the distributed environment, the ability to accurately assess the resource used generally comes down to measuring the CPU and memory consumed. In the WebSphere z/OS context, specific measurement of general loads should be done in conjunction with the available resources for that system and the z/OS Workload Management settings. In other words, a direct CPU usage measure will not provide a true measure of the workload a given transaction is consuming.

2.1.5 Availability

Availability in the infrastructure requires that the implemented configuration provide some hardware and software redundancy in order to eliminate single points of failure. In other words, both single points of failure from a system and an application server perspective should be avoided.

Redundancy can be implemented using two different techniques. Two systems or servers can be provided, each equally sharing the load, or a primary and backup, where the primary serves the required load and in the event of failure the backup automatically takes over.

Vertical scalability (cloning application servers within the same system) provides highly available application servers. However, the system that they run on then becomes a single point of failure.

To account for this, a highly available topology typically involves both vertical scaling within the system and horizontal scaling across multiple systems.

Hardware-based high availability

To provide continuous availability, the zSeries platform differentiates itself from other server platforms in terms of hardware and software reliability.

The zSeries server architecture includes self-healing attributes to prevent downtime caused by system crashes. Therefore, because of these functions, there is no outage in case of most hardware failures.

The operating system takes advantage of the self-healing attributes of the hardware, and extends them by adding functions such as recovery services for all operating system code, address space isolation, and storage key protection. Functions such as Workload Manager (WLM), Resource Recovery Services (RRS), and Automatic Restart Manager (ARM) assure the availability of applications.

To gain ultimate continuous availability with zSeries and z/OS, Parallel Sysplex clustering technology is available. This technology implements a data-sharing design that allows a database to be concurrently read and updated by application clones running on multiple z/OS images on one or more physical servers.

The Workload Manager balances application workloads across the systems in the sysplex. If there is a failure or a planned outage on one system, other systems within the sysplex will take over the full workload. By implementing Geographically Dispersed Parallel Sysplex (GDPS), the servers can be as far as 40 km from each other.

Failover

Providing multiple application servers on multiple systems provides for a natural failover capability. That is, if any one system or application server in the system were to fail for any reason, the system should continue to serve the overall client load on the remaining running servers.

The load-balancing property previously described ensures that the load gets evenly redistributed to the remaining active servers. These servers will each take on a slightly higher percentage of the total load on the infrastructure.

When designing this capability into the infrastructure, one must plan to allow the remaining available capacity, in the event of a failure, to serve the required load. In considering this eventuality, one must remember that should the capacity not be adequate, users will see a decay in response time. This decay may in fact be an acceptable outcome in the event of a system failure within an infrastructure.

This failover capability should be totally transparent to users of the infrastructure. When a failure occurs, the users that are currently directly interacting with the application server or systems concerned should be automatically redirected to one of the remaining servers/systems, without any obvious interruption of service and, more importantly, without requiring the user to do anything special.

In reality, however, failover may not be completely transparent to the user. For example, a client that is currently in the middle of an operation when a server fails may receive an error from that operation, and may be required to retry. This retry should connect the user to another server. The user may observe a delay in processing, before the processing of his request resumes automatically with a different server.

The important point in failover is that the user community as a whole should be able to continue with uninterrupted work in the event of a failure.

With applications requiring some state to be carried from request to request, providing transparent failover requires consideration to be given to the failover of state or user sessions.

When a previously failed server or system is brought back into service, the infrastructure should transparently start directing user requests to that server or system, ensuring that the workload is balanced across the infrastructure again.

The z/OS environment offers significant capability in this area and is considered a highly available infrastructure. The WebSphere environment that is deployed and “sysplex enabled” is naturally fault tolerant. This fault tolerance is due to the system capabilities of the z/OS platform. Consideration for state and/or user session failover still needs to be provided.

Looking in more detail at a specific application server: If a servant region fails, then other servants under the control region can still process work. If the control region itself fails and cannot be restarted on the same LPAR, then WebSphere z/OS provides Peer Recovery Restart functionality, which allows recovery in these situations and minimizes the impact of the failure.

2.1.6 Recovery

Resource Recovery Services

Many computer resources are so critical to a company's work that the integrity of these resources must be guaranteed. If the data in the resources is corrupted by a hardware or software failure, human error, or a catastrophe, the computer must be able to restore the data. These critical resources are called *protected* resources or, sometimes, recoverable resources.

Resource Recovery Services (RRS) is the IBM Transaction Manager on z/OS that coordinates changes to these protected resources. Subsystems such as DB2, CICS®, IMS™ and WebSphere MQ register with RRS as Resource Managers.

The application server itself is not a recoverable resource manager. It is a recoverable communication manager. It has no recoverable locks of its own and it doesn't need to manage locks or manage lock states in a log. It just needs to make sure that both callers and callees are connected in each of the communication sessions of a distributed transaction.

RRS coordinates the update of protected resources sysplex-wide by maintaining Units of Recovery (URs) in Coupling Facility Log Streams. These URs represent the state of a transaction, and allow protected resources to be “locked” until an outcome to the transaction is determined.

Peer Restart and Recovery (PRR)

When a system failure occurs, resource managers such as DB2 may have been processing transactions at the time. This may result in a situation where in-doubt URs are still held by RRS, effectively locking the protected resource until an outcome for the UR can be established.

It may not be possible to restart the application server on its own system, so WebSphere z/OS has a Peer Restart and Recovery (PRR) feature that allows the failed application server control region to be restarted on an alternate node in the same cell.

The control region will start in recovery mode and assign outcomes to all transactions that were in progress at the time of the failure. No data is lost during the process and all locks held by the URs are freed. During the recovery process the restarted server will not start up a Web or EJB container so that, for performance reasons, it will not take on any new work.

There are a number of prerequisites that need to be set up before PRR can be used. If you plan on using this recovery feature, then you should establish a process that can be easily followed during a failure situation. Refer to *IBM*

WebSphere Application Server for z/OS V5.0.2: Servers and Environment, GA22-7958 for prerequisites and more detail on the process.

XAResource Manager

In addition to RRS, many JTA XAResource Managers can be used to assist in a WebSphere Application Server for z/OS peer restart and recovery. If you plan to use WebSphere Application Server for z/OS peer restart and recovery, and your applications access JTA XAResource Managers, you must ensure that the appropriate logs and classes are available on the alternate system. The suggested place is a shared HFS.

High Availability Manager

A new component in WebSphere Application Server version 6 is the High Availability Manager, which provides peer-to-peer failover for critical services by always maintaining a backup for these services. It is responsible for managing the availability of *singletons* within the cell. These singleton services are important for recovery.

Examples of singletons include transaction managers for cluster members, messaging engines, and WebSphere workload manager routing information (WLM, which is different from z/OS WLM), etc. A High Availability Manager continually monitors the application server environment. If an application server component fails, the High Availability Manager ensures takeover for the in-flight and in-doubt work for the failed server. This action significantly improves application server availability.

2.1.7 Maintainability

Maintainability is the property that provides for simple management and change to be implemented within the infrastructure with minimum disruption.

There are specific issues that need to be considered when deploying a topology that is maintainable.

When implementing an infrastructure that is highly maintainable, one strives for simplicity. In the extreme case, this leads to only one application server, which comes into conflict with the principle of availability, which requires at least two application servers, ideally across more than one system.

Some of the maintainability aspects that we consider are:

- ▶ Dynamic changes to configuration
- ▶ Mixed configuration
- ▶ Fault isolation

Dynamic changes to configuration

In certain configurations, it may be possible to modify the configuration “on the fly” without interrupting the operation of the system and its service to clients. For example, it may be possible to add or remove servers to adjust to variations in the total client load. Or it may be possible to temporarily stop one server to change some operational or tuning parameters, then restart it and continue to service client requests. Such characteristics, when possible, are highly desirable, since they enhance the overall manageability and flexibility of the system.

The ability to make application deployments in WebSphere that are nondisruptive changes is an example of this dynamic change property.

Mixed configuration

The ability to mix configurations and, in some cases, versions, allows for smooth transitions from one version of WebSphere to another without disruption to service. When this capability is used in conjunction with the dynamic change capability, a new version of WebSphere may be “rolled” across the infrastructure. This, of course, requires careful planning and management.

Fault isolation

In the simplest failover scenario, we are only concerned with clean failures of an individual application server or system, where this failure has no effect on those application servers or systems around it, other than providing them with more load to handle.

However, there are some situations where one malfunctioning server may in turn create problems for the operation of other, otherwise healthy, servers. The easiest example of this is where one application server on one system starts to consume all available resources and makes no resources available to other application servers on the same system.

On z/OS such effects can be constrained to some extent. Workload Manager will determine the amount of resources allocated to an application server. Providing that the WLM definitions are sensible and well balanced, looping applications will not obtain resources from critical system functions.

2.1.8 Session management

Unless you have only a single application server or your application is completely stateless, maintaining state or session between HTTP client requests will also play a factor in determining your configuration.

In WebSphere V6.0 there are two methods for sharing sessions between multiple application server processes (cluster members). One method is to persist the session to a database. An alternate approach is to use

memory-to-memory session replication functionality. It is accomplished by the creation of a data replication service (DRS) instance in an application server that talks to other DRS instances in remote application servers.

The use of one or the other session failover capability should be carefully considered in light of the complete architecture.

2.1.9 Security

The potential to distribute the processing responsibilities between multiple servers and, in particular, multiple systems, also introduces a number of requirements on the security implementation. Different servers or types of servers may be assigned to manipulate different classes of data or perform different types of operations. The interactions between the various servers may be controlled, for example externally through the use of firewalls or internally through cross-server authentication, to prevent unauthorized access to data.

When deploying the WebSphere environment to z/OS, it can provide a highly secure environment where server and system interaction can be controlled by the onboard Security Access Facility (SAF).

2.2 Increasing performance

We now define what we mean by performance, how to measure and monitor it, and how to tune it.

2.2.1 Performance definition

When we refer to performance we usually think in one dimension: response time. Response time at any cost is an approach that has some valid business applications but in most cases we are concerned with two dimensions, response time and the resources consumed in providing that response time.

The key concern is how much hardware and system resource each individual transaction in each individual application costs to execute, that is, how efficient the program is, not just how fast.

If an application provides function X and in doing so uses 20% of the resources of a system, and the same function can be made to use only 10% of the resources of a system and provide the same or better response time—this has effectively halved that element of the cost of that transaction.

To be able to monitor and understand this in the J2EE context, the application server monitoring tool needs to be able to track the thread of execution through

the infrastructure and examine what resources this particular transaction is consuming. This capability is available in certain monitoring tools.

In the z/OS environment the monitoring of system resource consumption is normal daily business and as such the system keeps track of the “service units” consumed by an application server. Resource Monitoring Facility (RMF™) or WebSphere Studio Application Monitor can provide performance details for the application servers.

2.2.2 Performance measurement

Tuning parameter hot list

The following list of tuning suggestions is a subset of the complete list that follows in this chapter. These parameters are on a hot list because they have an important impact on performance.

The majority of the parameters are application-dependent, and should only be adjusted after close inspection of testing results for the application in question.

- ▶ Hardware and capacity settings
- ▶ WLM Service Class definitions, including response time goals and execution velocity
- ▶ Java virtual machine heap size and garbage collection parameters
- ▶ Application assembly performance checklist
- ▶ Data sources connection pool, prepared statement cache, and EJB Data Caching
- ▶ Network parameters
- ▶ Pass-by-value versus Pass-by-reference, including the use of EJB local and remote interfaces
- ▶ IBM HTTP Server settings
- ▶ Transaction logs

Parameters to avoid failures

The following parameters can create application failures and should be viewed as a priority. They can all be adjusted within the Administrative Console.

- ▶ Number of connections to a database.
- ▶ “Allow thread allocation beyond maximum” has been selected and the system is overloaded because too many threads are allocated.
- ▶ Connection Pool Size: Ensure enough connections for transaction processing with Entity EJBs and for avoiding deadlock.

For a detailed description of tuning for WebSphere z/OS, refer to *IBM WebSphere Application Server for z/OS V5.0.2: Performance Tuning and Monitoring*, SA22-7963.

2.2.3 Tuning performance

When faced with the need to increase performance, a number of reasonable techniques can be applied—all with varying possible results.

The possible paths are:

- ▶ Tune the application
- ▶ Tune the infrastructure
- ▶ Scale the infrastructure

The following section describes the first two techniques.

Tune the application

Tuning the application often provides the most significant increase in performance for the least cost, but is often the most difficult to understand and is the least used of the available techniques.

To best tune an application, a significant understanding of the application behavior in the runtime environment needs to be established. This behavior should include:

- ▶ The response time under single user load and under ever-increasing load
- ▶ The latency existing in any individual component under single user load and under ever-increasing load
- ▶ The cost in system resources under single user load and under ever-increasing load
- ▶ How the application interacts with other external entities, either other applications or other resources (for example, databases).

The acquisition of this information requires access to tools that operate in two environments.

The developer should be able to understand the runtime execution path of his application. WebSphere Studio Application Developer (WSAD) provides debugging and profiling capabilities that help the developer understand this.

The runtime should be analyzed under repeatable benchmarking conditions with visibility of the inside of the container. This visibility should be able to report not only the total response time, but also the response time of the individual

components, the resource usage within the application server, and the cost of the interaction with external resources. Tools such as the Tivoli® Performance Viewer and WebSphere Studio Application Monitor (WSAM) can provide this.

Armed with this information, the developer is now able to refactor his code to achieve a significant improvement in performance.

Focus areas

Developers should pay particular attention to the use of the services available with J2EE and the J2EE application programming patterns, the J2EE Core Patterns.

The use of these services and patterns allows the programmer to leverage the available capabilities of the WebSphere Application Server. These capabilities often provide highly efficient techniques that can have a significant impact on application performance.

There are numerous best practices documents around that clearly describe the techniques applicable in this area; these should be considered.

Advanced techniques

A number of advanced programming techniques can be used by developers to increase application performance. These techniques include using features available in the Programming Model Extensions to WebSphere Application Server. These programming models provide significant capabilities, but should be used with care.

If the application is not performing as desired or expected, initial focus should be on the use of the basic J2EE services and of correctly coded and relevant patterns. After this has been done, and performance is still not as desired or expected, then the application of more advanced programming techniques should be considered.

Tune the infrastructure

Having tuned the application and achieved close to the required performance, tuning of the infrastructure can now be undertaken. Tuning the infrastructure leads to a more efficient use of the available resources and allows the application to scale more naturally.

Note: Tuning of the infrastructure will not resolve application performance issues created by a poor application.

In tuning the infrastructure, particular attention needs to be paid to the resources that are made available by the application server and those needed by the application as the load increases.

WebSphere Application Server system queues

WebSphere Application Server allows a network of queues to develop in front of and within the application server. Queues develop at the entry point to the major components including the network, Web server, Web container, EJB container, Object Request Broker (ORB), data source, and possibly a connection manager to backend systems.

These queues are made up of requests waiting to have access to that resource. They develop when the number of requests coming into the system is greater than the number of available resources.

The user sees these queues as increases in response time to service requests. As they grow longer, the response times of all clients get slower.

As an example, think of an application consisting of servlets and EJBs that access a database. Each of these application elements resides in the appropriate WebSphere component and each WebSphere component can handle a certain number of requests in a given time frame.

A client request enters the Web server and travels through WebSphere components in order to provide a response to the client. Figure 2-1 illustrates the processing path this application takes through the WebSphere components as interconnected pipes that form a large tube.

This example is available in a number of other IBM high-volume Web site references and clearly shows the queuing in the application server.

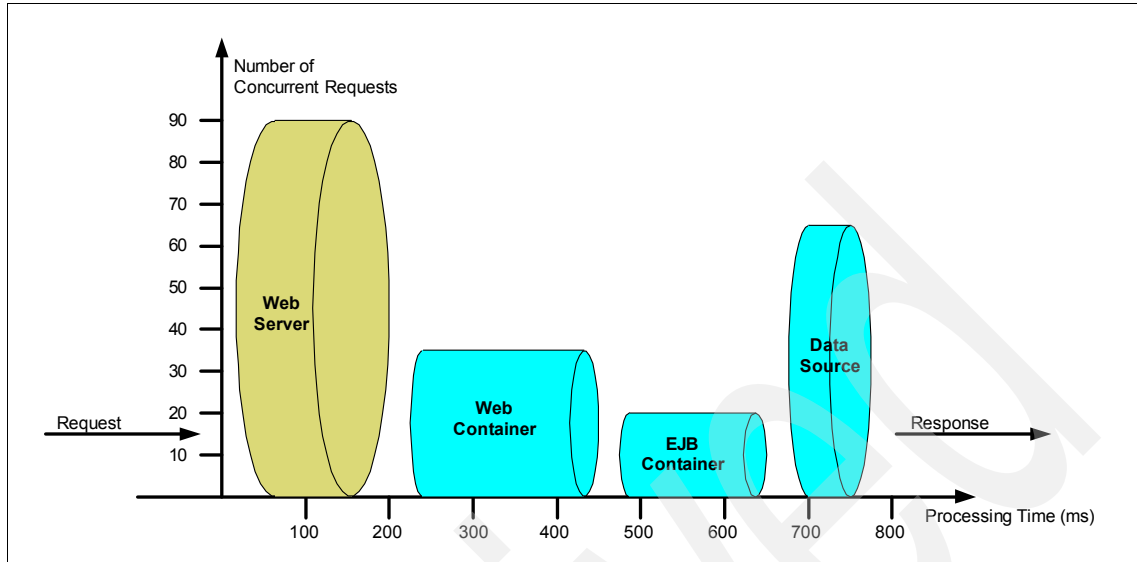


Figure 2-1 WebSphere internal queuing model

The width of the pipes (illustrated by height) represents the number of requests that can be processed at any given time. The length represents the processing time that it takes to provide a response to the request.

In order to find processing bottlenecks, it is useful to calculate an operations per second (tps) ratio for each component. Ratio calculations for a fictional application are shown in Table 2-1.

Table 2-1 WebSphere component throughput calculation example

Operation	Calculation	Total throughput
The Web Server can process 90 requests in 100 ms	90 requests/0.1s	900 requests per second
The Web container parts can process 35 requests in 200 ms	35 requests/0.2s	176 requests per second
The EJB container parts can process 20 requests in 150 ms	22 requests/0.125s	176 requests per second
The datasource can process 65 requests in 50 ms	65 requests/0.05s	1300 requests per second

The example shows that the application elements in the Web container and in the EJB container process requests at the same speed. Nothing would be gained in an application that uses EJBs from increasing the processing speed of the servlets and/or Web container because the EJB container would still only handle

176 transactions per second. The requests normally queued at the Web container would simply shift to the queue for the EJB container.

This example illustrates the importance of queues in the system. Looking at the operations ratio of the Web and EJB containers, each is able to process the same number of requests over time. However, the Web container could produce twice the number of requests that the EJB container could process at any given time. In order to keep the EJB container fully utilized, the other half of the requests must be queued.

It should be noted that it is common for applications to have more requests processed in the Web server and Web container than by EJBs and backend systems. As a result, the queue sizes would be progressively smaller, moving deeper into the WebSphere components. This is one of the reasons queue sizes should not solely depend on the operation ratios.

The following section outlines a methodology for configuring the WebSphere Application Server queues.

Queuing before WebSphere

In tuning the application server the goal is to minimize the number and size of the WebSphere internal queues. The ideal situation is for any queuing to occur in front of the Web server on the network.

When this is achieved, requests that can enter the infrastructure can be processed immediately. In creating this configuration, tune the relevant parameters to create a funnel with the larger number of resources available at the client and the lesser resources available near the core of the infrastructure.

When this configuration is implemented on one application server and when using a load balancing or workload management capability in front of this application server, queued requests can be directed to other more capable servers.

2.3 Scaling your environment

The following sections outline the techniques and methods that are applicable in the z/OS WebSphere Application Server environment and can be applied to effectively scale the infrastructure and achieve the objectives of scaling.

2.3.1 WebSphere scaling anatomy

Building upon the picture presented in Figure 1-2, there are some additional concepts in the WebSphere Application Server environment that are key to scaling the environment.

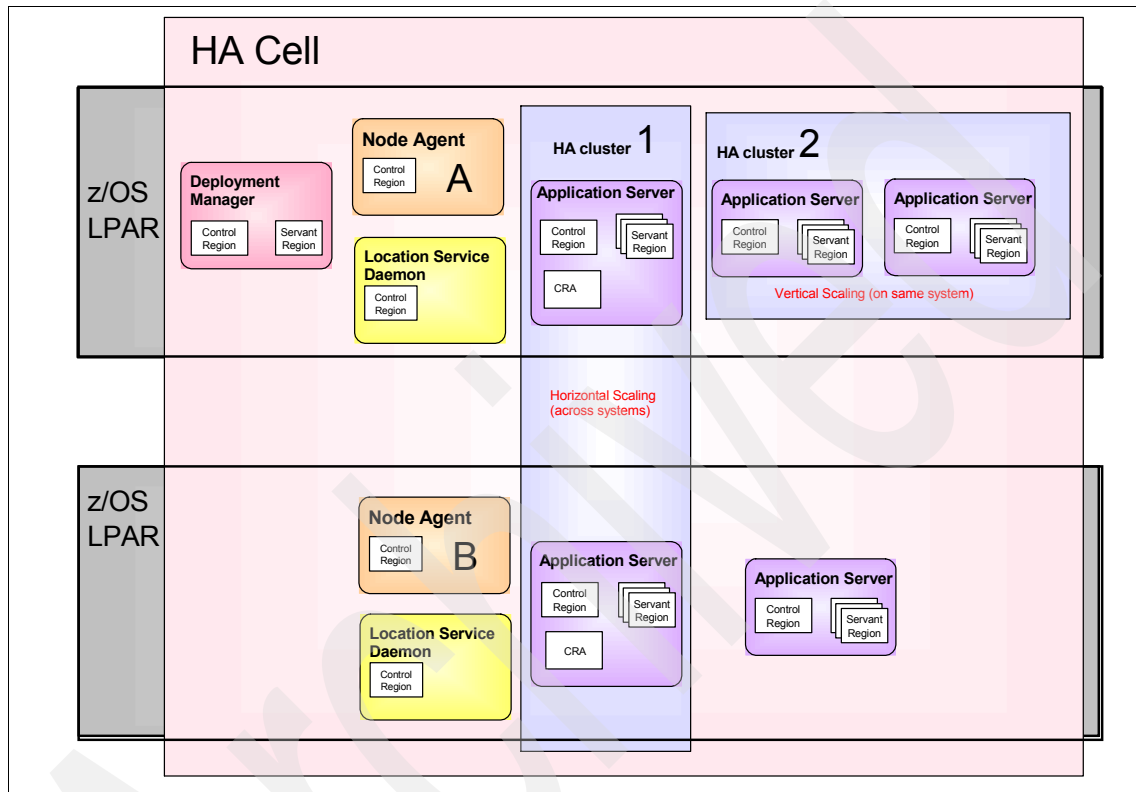


Figure 2-2 Scaling a z/OS WebSphere Application Server

Clusters and cluster members

A *cluster* is a logical grouping of replicated server instances, allowing you to partition your workload and still see it as one entity. Each of the servers is an exact copy of the others. The server instances belonging to a cluster can run on the same z/OS image, or can span more than one image in a sysplex.

From the client point of view, the cluster is an entity, and the client should not know which server instance will perform the given request. So a cluster is a set of application servers that are managed together and participate in workload management, and can be considered a single application-serving unit from a client perspective.

In the WebSphere Application Server for z/OS V6 environment, only systems of the same operating system type can exist within the same cell.

The major constraint for cluster members is that they are required to have identical application components, while these can be sized differently in terms of weight, heap size, and other environmental factors. The fact that environmental settings for each application server can be different allows the cluster to run across a range of z/OS servers of different capabilities.

Administrative activities can be performed on the cluster as a whole or on individual servers.

Figure 2-2 on page 34 shows an example of a possible configuration that includes application server clusters. Server cluster 1, which is completely independent of server cluster 2, has two cluster members, one on each node. Server cluster 2 has two cluster members, both on node A. Finally, node B contains a free-standing application server that is not a member of any cluster.

Cells

A cell is a collection of application servers on one or more nodes. It may contain clusters or individual application servers or a mixture of both. The cell is the outermost entity that is the administrative domain of WebSphere.

There is one Deployment Manager per cell and that is what is used to manage all the nodes within a cell. The configuration and application byte code of all nodes in the cell is centralized into a cell master configuration repository. This centralized repository is managed by the Deployment Manager process and synchronized out to local copies held on each of the nodes.

2.3.2 Scaling process

Clustering is an effective way to perform scaling of application servers. This scaling can be done both vertically and horizontally.

Vertical scaling

On z/OS, within an application server, the WorkLoad Manager will start as many servant regions as required (within imposed restraints) to process the workload and meet the defined goals. If a given server is overloaded, it is temporarily bypassed in favor of less busy servers. If a server fails, other servers take over the work and the server is recovered. When the servers are no longer needed, they are automatically stopped.

This kind of automatic vertical scaling allows the full capabilities of z/OS workload management to be enabled. In the non-z/OS environment, the clones need to be preconfigured and are not dynamically created.

Alternatively, you can add more control regions into the WebSphere cluster on the same LPAR or node as shown in HA Cluster 2 in Figure 2-2 on page 34.

This is not normally recommended as a way of increasing performance unless the control region is becoming the bottleneck to the servant regions and cannot queue requests to WLM fast enough for the servant regions to process.

Note that for reasons of high availability, even if a single JVM can fully utilize the processing capability of the node, you may want to have more than one cluster member on the node for software failover. If a JVM reaches a table or memory limit (or if there is some similar problem), you can have another Application Server process to go to for failover.

So while this configuration is not recommended as a way of increasing performance, it can be desirable in terms of high availability.

Note: The initial minimum number of servant regions defined per LPAR should be at least one more than the number of Service Classes that are defined for the application server in the Workload Manager policy.

The control region queues requests to WLM to be processed by the servant region that represents the Service Class of the request. Therefore, you need to have enough servant regions to process these requests (and default service class requests).

To establish the maximum number of servant regions, you must take into account the memory that you actually have on the LPAR. Each servant region will take at least 200 MB. You should increase the maximum number of servant regions in a systematic manner, measuring memory usage and performance at each step.

Hardware limitations aside, the addition of each servant region should, in fact, increase throughput under maximum load conditions. If the addition of a servant region does not achieve this, then that additional servant region is not required. If this is the case, then one should reexamine other bottleneck areas, including the control region and the application.

Note: Always monitor memory usage when you are configuring a vertical scaling topology.

Horizontal scaling

Horizontal scaling is especially effective in environments that contain many smaller, less powerful systems/nodes. Client requests that overwhelm a single system can be distributed over several systems.

Failover is another benefit of horizontal scaling. If a system becomes unavailable, its workload can be routed to other systems containing cluster members.

The use of horizontal scaling with WebSphere on z/OS is the natural deployment configuration that can best take advantage of the features of z/OS.

HA Cluster 1 in Figure 2-2 on page 34 is an example of horizontal scaling, where the two clusters are spread across different LPARs.

More systems and/or nodes can be added to the infrastructure to allow the workload to be spread more widely. In zSeries terms this could mean adding more LPARs to the configuration of a single machine, or adding more machines to a sysplex.

Parallel Sysplex architecture is designed to integrate up to 32 systems in one cluster. The sysplex clustering technology (this is what we refer to here) is not the same as WebSphere clustering, although the goals are similar. WebSphere for z/OS exploits sysplex clustering. Each of these systems is able to handle multiple different workloads and access databases using data sharing technology.

With zSeries, a function called Intelligent Resource Director (IRD) reassigns resources on demand to different logical partitions. Parallel Sysplex architecture provides performance for workloads with unpredictable demands: It can run WebSphere and traditional OLTP/DB applications simultaneously at 100% utilization. More work is processed within a single server, without over-configuring for complementary peaks.

Horizontal scaling can handle application server process failure and hardware failure without significant interruption to client service.

WebSphere applications can combine horizontal and vertical scaling to reap the benefits of both scaling techniques.

The only way to determine what is correct for your environment and applications is to tune a single instance of an application server for throughput and performance, then add it to a cluster, and incrementally add additional cluster members.

Test performance and throughput as each member is added to the cluster. As each cluster member is added to the system, this should allow the cluster to achieve higher throughput. If this is not achieved when a cluster member is added, the previous number of cluster members was the optimal number for the configured infrastructure.

2.3.3 Understanding your workload

To establish what scaling techniques to apply to the infrastructure, one must understand what pieces of the infrastructure need to be scaled. To achieve this, one must understand the profile of the workload generated on the infrastructure itself.

When designing an infrastructure that offers scalability, high availability, performance, and manageability, this technique will provide a starting point and indicate the techniques that should be most applicable in scaling the infrastructure.

Base components

The following components are some of the basic elements that make up a WebSphere infrastructure:

- ▶ Edge Cache and Load Balancer
- ▶ HTTP server and WebSphere plug-in
- ▶ Web container
- ▶ EJB container

In placing these basic components on a system, particularly a single z/OS, we must be aware that these components

- ▶ Compete for machine resources.
- ▶ Influence each other's behavior.

One strategy is to physically separate some components, preventing them from competing for resources (CPU, memory, I/O, network, and so on). In a z/OS environment this is not necessarily a practical solution. The more efficient strategy is to use scaling techniques in conjunction with workload management, including horizontal and vertical scaling.

Business model, workload pattern

We know that different business patterns provide different workloads for the different components in an infrastructure. How the workload patterns use the different elements of the hardware as a percentage of the total workload is shown in Figure 2-3.

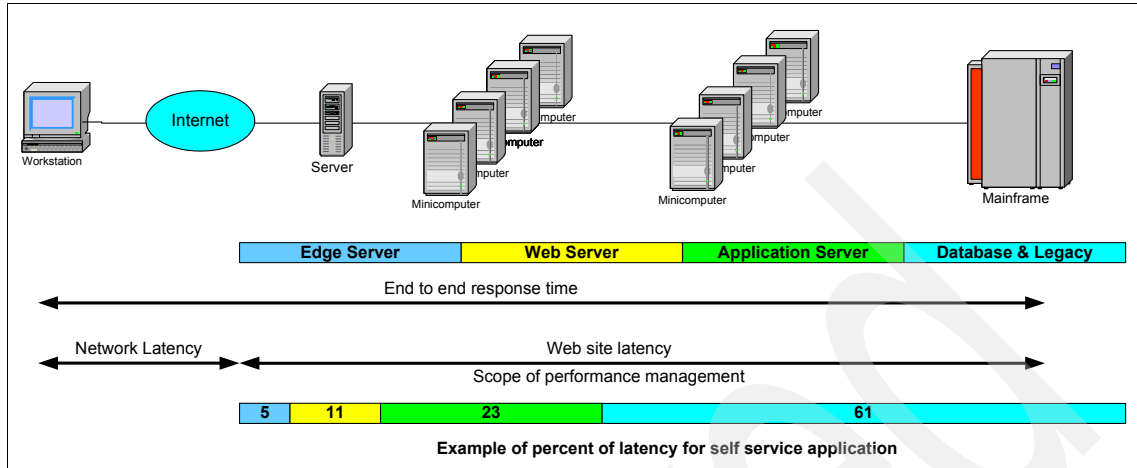


Figure 2-3 Workload pattern across WebSphere infrastructure

Each workload pattern has a specific characteristic that is built from a number of scenarios. Table 2-2 describes a number of scenarios that could exist, and the number of page views for each scenario.

Table 2-2 Workload scenario overview

Scenario	Percentage	Page Views
Search	85%	2
Update	15%	4

To further refine the model, additional factors are required.

The think time is the delay that exists for a user receiving a page and requesting the next page. In general, a reasonable estimate is 30 seconds. This figure has a significant impact on the number of concurrent users; the larger the think time, the longer the user sessions are. The increased number of concurrent users increases the requirements for system resources, for example memory, HTTP sessions, and TCP connections.

The *burstiness* is used to represent the fact that requests do not arrive at the infrastructure in a uniform distribution, they tend to arrive in close groupings, that is, bursts. This burstiness affects response times because it applies an uneven load on the system.

User behavior

There are a number of terms that help us define user behavior, as follows:

- ▶ Response Time - The time from submitting the request from the client to the complete response being displayed on the client
- ▶ Throughput - The amount of data being passed over the network in Kilobytes per second
- ▶ Hits Per Second - The number of individual HTTP requests being sent from the browser and being received by the server one second (number of pages x number of items on the page)
- ▶ Page Views Per Second - The number of complete pages that are displayed by the infrastructure in one second
- ▶ Transactions per second - The number of identifiable pieces of work per second, which can be viewed from a business or technical perspective from the front or back end.
- ▶ Concurrent Requests - The number of requests that can be simultaneously handled by the infrastructure
- ▶ CPU Consumption - The amount of CPU consumed while executing either a single request or multiple concurrent requests
- ▶ Virtual Users (VUsers) - The number of simulated users provided by tools such as WebSphere Studio Workload Simulator or Mercury LoadRunner (These can be run on one or many systems.)
- ▶ Think Time - The time interval that exists between each request sent to the server from a client perspective

From the above information we can now start to build a view of how the load is built up, when it occurs, what pieces of the infrastructure are loaded in what way—and from this data design an infrastructure that can satisfy the specific needs.

In addition we can establish some “What if” scenarios to understand what the future infrastructure would look like, based on various growth scenarios.

Infrastructure model

The hardware model provided by WebSphere can be either 1, 2 or 3 tiers with Web servers, application servers, and database servers. The software is assumed to consist of servlets/portlets, optional EJBs, databases, optional persistent sessions, and the use of SSL.

In building an infrastructure, optimize it for scalability. To achieve this, you should:

1. Thoroughly understand the application environment.
2. Categorize the workload.

3. Determine the components most impacted.
4. Select the scaling technique to apply.
5. Apply the scaling technique.
6. Review performance.

2.3.4 Scaling techniques

When the complete picture of the workload has been established, then the applicable scaling techniques can be selected and matched to the components concerned.

When developing and applying scaling techniques you should avoid compromising manageability, security, and availability. It should be evident that the introduction of scaling will increase complexity, and therefore manageability needs to be carefully considered.

Following are the eight commonly used scaling techniques:

- ▶ Use a faster system
- ▶ Create a cluster of systems
- ▶ Use appliance servers
- ▶ Segment the workload
- ▶ Batch requests
- ▶ Manage connections
- ▶ Cache

For application servers, the main technique for the growth path is to add more systems or to add more capacity to existing systems. It is therefore appropriate to start with the expectation of having more than one application server with a workload management or workload distribution component such as Load Balancer from Websphere Edge Component or Sysplex Distributor in front. If you start with this expectation, then adding more systems becomes easy and far less disruptive.

Using a faster system

The use of a faster system applies to the dispatchers or Edge Servers, the Web presentation server, the Web application server, the directory and security servers, the existing transaction and data servers, the network, and the Internet firewall.

The goal is to increase the ability to do more work in a unit of time by processing tasks more rapidly. Upgrading the hardware or software will result in a faster

system. However, one of the issues is that software capabilities can limit the hardware exploitation and vice versa. Another issue is that changes may be needed to existing system management policies.

The latest z990 series systems have been specifically designed to increase the capacity and throughput by providing specific services aimed at creating a high performance J2EE platform.

Creating a cluster of systems

Creating a cluster of systems applies to the Web presentation server, the Web application server, and the directory and security servers. The primary goal here is to service more client requests. Parallelism in system clusters typically leads to two effects: an improvement in response time, and the ability to serve more concurrent requests and maintain the same response times for all users. Also, system availability is improved because of the failover capability.

The service running in a clone may have associated with it state information that must be preserved across client requests, and this needs to be considered.

Two approaches to this problem exist, affinity and sharing state information among systems. This is discussed in Chapter 4, “HTTP sessions” on page 69.

Using appliance servers

This technique applies to the Edge Servers, the Web presentation server, the directory and security servers, the network, and the Internet firewall.

The objective of using appliance servers is to create functionality-based servers that have been tailored and tuned to a specific application. This technique is applicable where elements of the application have specific workload profiles that are outside the normal application profiles.

For example, in an e-commerce site that has the purchasing function concentrated because of the high transaction nature against the rest of the site, which is browsing intensive, this may mean having “purchasing” servers.

There are some additional considerations when using appliance servers. The splitting of an application may require additional, more complex, management, security, and deployment capabilities.

This technique is similar to workload segmentation, although in this case specific application functionality is the basis for separation, not a general workload profile.

Segmenting the workload

This technique applies to the Web presentation server, the Web application server, the data server, the intranet firewall, and the network.

In segmenting the workload, the objective is to break down the overall workload into segments that can be more easily tuned and therefore provide more consistent scaling and response times. The use of this technique allows the workload to be more effectively managed and scaling techniques applied to different elements of the workload in a more specialized way.

To achieve workload segmentation, an understanding of the workload profile is paramount. This technique may also require more complex management and security mechanisms.

The application of this technique fits closely to the tiers available in the J2EE architecture. Often the segmentation is directly along these tiers, that is, the presentation and business tiers are separated.

Batch requests

This technique applies to the Web presentation server, the Web application server, the directory and security servers, the existing business applications, and the database.

The main objective of this technique is to reduce the number of requests that are sent between the different tiers or parts of the infrastructure.

This technique is applied to deal with the use of remote interfaces to EJBs where the presentation services are running on one tier and the EJB components on another.

Fine-grained calls to each of the operations on a remote object increase the workload and cross-network communication. The use of a data transfer object (DTO) and a façade can then create a single cross-network call and allow the façade to make repeated calls to local EJBs that implement the required functionality and then return a single DTO to the client. See 2.5, “Application programming patterns” on page 47.

Managing connections

This technique applies to the Web presentation server, the Web application server, and the database.

In running applications that require communication between different components of different systems or subsystems, establishing a connection between these components has a significant overhead. To reduce this overhead the infrastructure should be configured with pools of ready-to-use connections

that the applications can call on when required. These connections can then be shared between the application requests, removing the need to establish these connection on an as-needed basis.

WebSphere Application Server has connection pooling techniques that are applicable to nearly all J2EE resources, so not only can connections to databases be managed, but connections to existing resources using the Java Connector Architecture can be connection-pooled as well.

Specific security and identity considerations need to be addressed when different users are potentially using the same database or connector.

The use of this technique is one of the key tools in maintaining and providing application scalability.

Caching

Caching applies to the dispatcher or Edge Server, the Web presentation server, the Web application server, the network, the existing business applications, and the database.

The objective of caching is to provide a funnel from the user to the deepest part of the infrastructure where the number of requests passing through each layer is steadily reduced till only those requests that must be processed by elements deep in the infrastructure are sent to that part of the infrastructure. In essence this technique is about offloading requests to be processed closer to the user.

The result of this technique is improved response time for the user and a reduction in the CPU used by the user community.

Caching techniques can be applied to both static and dynamic pages. In identifying those dynamic pages that have a life cycle that allows them to be cached, the application may have to be slightly modified to be able to isolate these pieces of content. In addition, even if a piece of data has a short life of only 30 seconds, in a high volume situation this may make a significant difference.

2.4 Increasing throughput

Increasing the throughput of a system or infrastructure means doing more with the same. Increasing the number of requests that can be served without increasing the physical capacity of the infrastructure improves the return on investment of the deployed infrastructure.

Performance tuning and scaling provides some of the desired results. However, increasing the throughput can be achieved by changing the server that responds to the user request and not executing the full request.

This outcome requires the use of caching and serving multiple requests using a previously garnished result.

The throughput of an application server is also dependent on the amount of time the Java Virtual Machine spends processing user requests against processing its own life cycle, specifically garbage collection.

2.4.1 Caching

As the use of Web site and applications running in application server environments, the users expect the response times to remain consistently good, and the business or organization providing the application expect to be able to serve increasing numbers of users of potentially the same infrastructure.

These expectations, however, involve contradictory requirements: On the one hand, more hardware needs to be provided to serve ever-increasing numbers of users with the same level of responsiveness, and on the other hand, more and more users need to be served with the same resources.

In addition, the users' expectation of content and personalization has increased, where sophisticated user interfaces are expected.

These conflicting requirements and user expectations increase demand on both network and application servers, creating bottlenecks and eventual degradation of performance.

Caching capabilities have been used for some time to improve performance. These common techniques include browser, Web server and some basic server side caching primarily focused on *static* content.

The greatest opportunity for caching can be gleaned from focusing on the caching of *dynamic* content. Content that is deemed dynamic is that which changes or is generated often and is potentially highly personalized.

The caching of dynamic content adds significant complexity to defining when and how the cache is to be invalidated. These mechanisms can include:

- ▶ An event based on user activity
- ▶ An event based on some server or content provider activity
- ▶ Time-based

The use of caching technology, which improves the return of investment on the base infrastructures, is critical in today's high scalability environments.

Overview

The use of caching can improve response time and reduce system load, enabling the existing infrastructure to serve more requests.

Most modern applications are initially considered as not cachable. This view is in most cases an over-generalization. There is data in most applications that can be cached; the important fact is to establish what can be cached and over what period the cached item is valid.

WebSphere Application Server provides a number of built-in dynamic cache services for serving dynamic content and caching data. These caches include:

- ▶ Dynacache - JSP™, servlet and portlet caching within the application server
- ▶ EJB Data Cache - Caching for CMP-based Entity EJBs in the application server
- ▶ Prepared Statement Cache - Caching of BMP-based SQL statements in the application server
- ▶ Fast Cache Accelerator - Static content cache on the HTTP Server
- ▶ Edge Cache - Static and dynamic cache at edge of network using Edge Components
- ▶ Command Cache - Programmatic cache based on Command Pattern in the application server
- ▶ Distributed Map - Application programmable caching interface in the application server
- ▶ JNDI name cache - WebSphere by default caches all objects looked up from WebSphere JNDI namespaces. The cache can be programmatically controlled.

These caches are primarily enabled or disabled declaratively using XML configuration files or by configuring EJB deployment descriptors. These techniques allow caching to be enabled quickly and effectively.

There are a number of programmatic capabilities that can be enabled in applications that make use of standard infrastructure capabilities.

Furthermore, one can use edge-of-network or Web server capabilities that can be effectively integrated into the application server environment

2.5 Application programming patterns

When developing an infrastructure that supports a range of requirements, it is important that the programming patterns that are used in this infrastructure are in accordance with the requirements and leverage the capabilities of the environment being deployed.

In the J2EE programming environment, there are a number of “core patterns” available. A subset of these can be used to effectively leverage the capabilities of WebSphere, such as:

- ▶ Naming and lookup
- ▶ Workload Management and failover
- ▶ Deployment
- ▶ Caching

The following section describes some of the core patterns and their relevance to the WebSphere infrastructure.

2.5.1 Applicable patterns

The following is a basic description of the core patterns that are relevant. These are derived from the Sun J2EE Core Patterns documentation.

Business Delegate

The use of the Business Delegate pattern reduces the coupling between presentation tier and enterprise components. The Business Delegate hides the underlying implementation details of the business component and business service such as lookup and access details of the EJB architecture.

This pattern is particularly important where access to legacy of enterprise systems is required as this avoids coupling the client application and the backend applications together.

Session Facade

Use a session bean as a facade to encapsulate the complexity of interactions between the business objects participating in a workflow. The Session Facade manages the business objects and provides a uniform coarse-grained service access layer to clients.

The Session Facade offers strongly typed public method representations of the services they offer.

The Session Facade should exist in the case where it actually implements the business flow of the required service or acts as a proxy to a service on some non-J2EE enterprise system or legacy platform. This session bean encapsulates the access to this service that may be through J2C or over MQ Series or some other transport mechanism.

The implementation of this session bean for a J2C-connected non-J2EE enterprise system allows the service to move from legacy to J2EE without impacting the Business Delegate.

Service Locator

Use a Service Locator object to abstract all JNDI usage and to hide the complexities of initial context creation, EJB home object lookup, and EJB object recreation. All clients that use a Business Delegate use the Service Locator by default. The Service Locator has been included in the Business Delegate to reduce code complexity, provide a single point of control, and improve performance by providing a caching facility.

Note: Service Locator patterns often include implementations for caching JNDI objects that they look up. With WebSphere V6 this functionality is not required since the default behavior of the Naming Service is to cache all JNDI lookups.

Data Transfer Object

Use a Data Transfer Object to encapsulate the business data. A single method call is used to send and retrieve the Data Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Data Transfer Object, populate it with its attribute values, and pass it by value to the client.

In a distributed deployment environment and with the use of the remote interface, the developer does not know whether a component is local to the next component or not. The remoteness can create a large and unnecessary network overhead. The use of a DTO allows a single remote method call to be used, that is, coarse-grained calls.

The use of DTO also allows the inclusion of information required by the legacy environment without making this visible to the developer. This information may be added when the object is instantiated through the object constructor.

2.5.2 Other considerations

Configured bindings

The use of configured bindings allows the actual reference to the installed component to be shielded from the client. The client references the configured

binding stored in the cell or persistent segment of the WebSphere namespace, and this provides the direct reference to the installed component.

This decoupling of the client from the actual deployed location of the component allows the component to be redeployed in a different location without impacting the client application.

Strongly typed interfaces

The use of a strongly typed interface on the components both at the EJB and Business Delegate level allows the generation of Web services definitions directly from the interfaces. It only provides visibility of the business contract provided by the service.

In addition, the use of strongly typed interfaces allows the use of technologies such as Process Choreographer available in WebSphere Application Server Enterprise Edition.

Business Delegate Factory

The use of a Business Delegate Factory allows the provision of a number of Business Delegate implementations and the use of these implementations to be configured by the operations staff, as opposed to these decisions being made by the developer and then having deployment changes impact the application.

The implementation that should be available includes:

- ▶ Business Delegate Local - Supporting the local interface to the EJB
- ▶ Business Delegate Remote - Supporting the remote interface to the EJB
- ▶ Business Delegate Fake - Allowing clients of the Business Delegate to develop and test against it

Other different Business Delegates can then be added as demand requires. Examples include a Business Delegate Cachable that caches the returned DTO for subsequent use.

Remote and local interfaces

Since EJB components use both local and remote interfaces, each session bean should be developed with both and deployed with both. This transfers the decision of whether to use local or remote to operations staff and away from developers. The operations staff then has the ability to switch either based on deployment or performance requirements without impacting the development teams.

J2EE Connectors

The use of the J2EE Connectors Architecture provides a standard way for all J2EE applications to communicate with other legacy environments. J2CA provides the standard Common Client Interface for all clients to use.

The connector itself implements the transaction, security, and connection management contracts with the container and there hides all the implementation specific to the remote platform and provides the J2EE developers with a J2EE resource as the means to access complex legacy environments.

Caching

Developing applications where some of the dynamic content is cachable requires some thought. While the starting and stopping of caches is configurable in the WebSphere runtime, developers should consider some factors.

In providing caching capability of the user interface, developers should consider fragmenting the page into cachable and non-cachable fragments. The page is broken down into fragments, where the fragment ties to a particular execution path through the infrastructure where infrastructure caches can then be used, therefore removing the need to develop caching mechanisms.

The use of servlet caching is fairly standard, but in the portlet environment where a number of portlets are on the same page, in the page generation some portlets could in fact be cachable. For portlet development refer to the WebSphere Portal Server documentation.

The use of the command cache allows the execution result of a component to be cached for future use. This could be applied to stateless session beans, where the execution results are reusable.

Finally, the standard caching infrastructure can be used through the Distributed MAP programming model. The development of a Business Delegate that passed a DTO to a session bean and received a DTO as a result could use Distributed MAP to implement a caching mechanism for the DTO, which would allow the DTO result set to be used again and again without the application having to maintain state.

Stateless applications

The use of stateless only applications allows the effective workload management of requests to be transparent. This allows the use of the instance-pooling mechanisms available in WebSphere to create and store a number of instances of the components, and these components can then be served to any inbound request requiring that particular component.

The drive to stateless applications is contrary to the basic conversation state required of most modern applications. This state should be stored in the WebSphere HTTP Session. This session service then provides persistence and failover through a number of different mechanisms. Large HTTP Session objects should be avoided; other mechanisms such as caching and persistence should be considered.

Archived

Archived



Workload balancing

In this chapter, we describe workload balancing and explain its interrelationship with high availability. We also look at Domain Name Service, Edge Server, Sysplex Distributor and MNLB solutions as they apply to the z/OS environment. We also briefly discuss interactions with third-party dispatchers like Cisco's CSS switch.

3.1 Workload balancing

Workload balancing is the ability to distribute workload to the available computer resources in a way that maximizes the utilization of these resources. There are a number of mechanisms for balancing workloads. Figure 3-1 shows five logical functions of an e-business infrastructure: Edge servers, Web and Web application servers, directory and security servers, data and transaction servers, and storage systems. We can balance the workload in front of each set of servers.

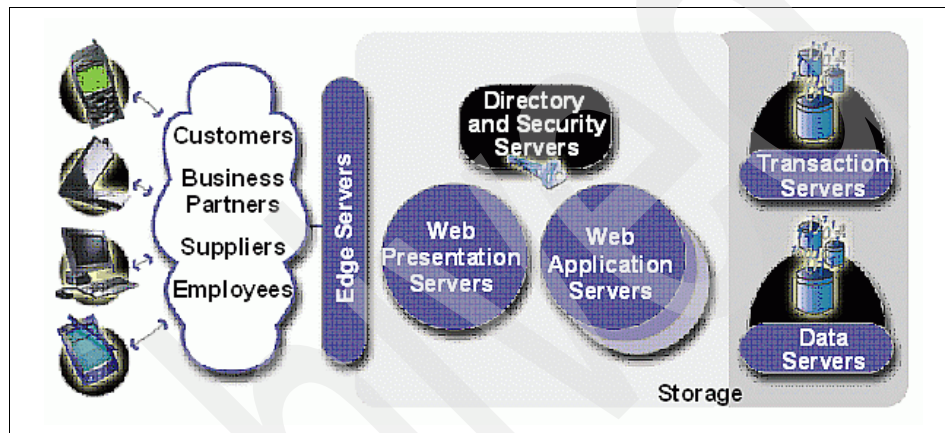


Figure 3-1 Five logical functions of an e-business infrastructure

We focus our attention on the first three servers: Edge, Web, and Web application, as well as Security servers.

Workload balancing of incoming requests (or network balancing) ensures that such a group (or cluster of servers) can maintain optimum performance by serving client requests simultaneously. Additionally, an evenly spread network minimizes the number of users affected by the failure of a single server. Thus, network balancing and availability are closely linked.

Clustering techniques that are providing network balancing must also provide for other system requirements in addition to the dispatching of connections. These include the ability to advertise some single system-wide image or identity so that clients can uniquely and easily identify the service.

Typically, this system-wide identity is either an IP address, known as the *cluster address*, or a host name, known as the *service name*.

On z/OS, VIPA TCP/IP addresses are used to provide automatic backup and routing via the Sysplex Distributor. See 3.3.2, “Sysplex Distributor” on page 62 for more information.

In the case of cluster addresses, clients will always use the service via the same IP address. In the case of service names—although the host name is always the same—the service will be identified by different IP addresses, depending on server load.

This leads to two categories of clustering load distribution solutions:

- ▶ DNS mapping
This is not a strategic form of load distribution.
- ▶ Connection dispatching
This is a more flexible and robust method, which we used in our architecture; see 3.3, “Connection dispatching solutions” on page 58.

3.2 Domain Name Server mapping solutions

Domain Name Server (DNS) mapping refers to the dynamic changing of DNS entries to map a service name (or host name) to different IP addresses in order to identify which target server should receive new connections. This straightforward mechanism has its limitations, including its dependence on host name resolution for load distribution. As a result, it is *not* a strategic form of load distribution.

On the z/OS platform, there are two DNS mapping solution options, neither of which is considered strategic, but which are in use today in many customer environments. We therefore describe them here briefly for completeness:

- ▶ Round-robin DNS
- ▶ Connection Optimization (DNS/WLM)

3.2.1 Round-robin DNS

Early solutions to address load balancing were often located at the point where host names are translated into actual IP addresses. By rotating through a table of alternate IP addresses for a specific service, some degree of load balancing is achieved. This approach is often called *round-robin DNS*. The advantages of this approach are that it is transparent both to the client and the destination host, and that it is performed only once, at the start of the connection.

Figure 3-2 shows an example of round-robin name resolution. In this case, the client resolves the name “www.redbooks.ibm.com”. The DNS server responsible for this domain cycles through its list of possible target servers and returns to the client the next server to use.

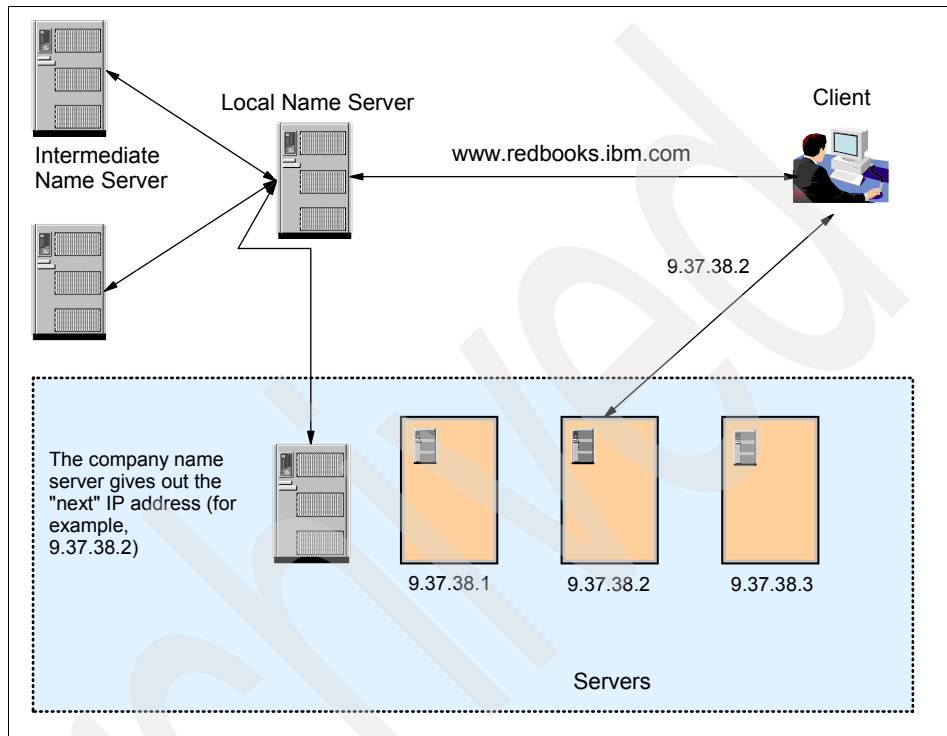


Figure 3-2 Round-robin DNS

Unfortunately, this approach is sometimes defeated by the fact that intermediate name servers and client software (including some of the most popular browsers) cache the IP address returned by the DNS service and ignore an expressly specified time-to-live (TTL) value of the name resolution.

As a result, the balancing function provided by the DNS is bypassed because the client continues to use a cached IP address instead of resolving again. Even if a client does not cache the IP address, basic round-robin DNS still has other limitations:

- ▶ It does not provide the ability to differentiate by port.
- ▶ It has no awareness of the availability of the target servers.
- ▶ It does not take into account the workload on the target servers.

3.2.2 Connection optimization (DNS/WLM)

This DNS solution is based on the DNS name server and the z/OS Workload Manager.

Intelligent sysplex distribution of connections is provided through cooperation between WLM and DNS. For customers who elect to place a name server in a z/OS sysplex, the name server can utilize WLM to determine the best system to service a given client request.

In general, DNS/WLM relies on the hostname-to-IP address resolution for the mechanism by which to distribute load among target servers. Hence, the single system image provided by DNS/WLM is that of a specific hostname.

Note that the system most suitable to receive an incoming client connection is determined only at the time of hostname resolution. Once the connection is made, the system being used cannot be changed without restarting the connection.

This DNS approach works only in a sysplex environment, because the Workload Manager requires that environment. If the server applications are not all in the same sysplex, then there can be no single WLM policy and no meaningful coordination between WLM and DNS. In brief, it functions as follows:

- ▶ The servers register with WLM under a particular *server group name* for the purpose of providing some service described by this name (for example, TN3270).
- ▶ At regular intervals, DNS asks WLM for its workload measurements.
- ▶ The client requests a server by IP hostname via a process called *hostname resolution*. This hostname is the server group name known to WLM.
- ▶ The DNS name server in the sysplex checks its WLM information for details of the servers registered under that host (server group) name.
- ▶ DNS responds to the client with the IP address of the most suitable server instance so that the client can connect to that specific server.

All BIND-based name servers use a simple sorting algorithm (by default) when returning one of multiple addresses during hostname resolution, and addresses obtained via DNS/WLM are no exception. This is more or less the basic round-robin technique.

However, WLM also has some additional notion of performance or server load, and can return addresses for the server with the least current load. As a result, we get a lot more than just simple round-robin selection. WLM will provide host and application weights that help the name server to load balance the connections to the sysplex servers.

If the application is registered to WLM, the name server will resolve the client's name query for the application; and if the application is de-registered, WLM will no longer provide its details to DNS. This means that dead applications will not be selected by DNS.

This solution is only available with the BIND 4.9.3 name server and not with the BIND 9 name server. Also, as mentioned, the DNS approach works only in a sysplex environment, because Workload Manager requires it.

Once again, neither DNS solution is considered strategic.

3.3 Connection dispatching solutions

Connection dispatching is the routing of TCP connections from a dispatching (or distributing) node to a group of target servers. With this technique, the client perceives a traditional TCP connection with a server. The dispatching node, however, receives data from the client and forwards it to the appropriate server, which can reply directly to the client.

All systems in this cluster provide information about their workload to a dispatching entity, which is generally referred to as a *distribution manager*. This manager is responsible for distributing connection requests from clients to the target systems where the application servers are running. The distribution is based on the current workload information collected by the distribution manager.

Users at client workstations are not aware of such application server clusters. They try to connect to a service with a virtual IP address, assuming it is running in the machine of the distribution manager. Therefore, the name server should translate the cluster name of the application servers into this virtual address.

The z/OS platform (as a target server) has a number of options of this type including:

- ▶ Load Balancer
- ▶ Sysplex Distributor
- ▶ Sysplex Distributor with MultiNode Load Balancing (MNLB)

3.3.1 Load Balancer

The WebSphere Edge Component includes the Caching Proxy and Load Balancer. There are well-written descriptions of the components in the WebSphere Edge Components InfoCenter at:

<http://www-306.ibm.com/software/webservers/appserv/ecinfocenter.html>

The proxy server can accept requests from a client (or load balancer) and proxy or cache the request for later requests of the same nature. In our setup, we installed the proxy server just to show how to do it, but have not really used it. Our application was much simpler and did not really exploit the proxy server.

The Load Balancer distributes the load between multiple clustered servers (any type of TCP/IP server). Depending on the type of traffic you want to load and how this will happen will affect the type of load balancing you use.

There are five individual components that come with the Load Balancer:

- ▶ Network Dispatcher
- ▶ Content Based Routing
- ▶ Site Selector
- ▶ Cisco CSS Controller
- ▶ Nortel Alteon Controller and the Metric Server

In our configuration we used the Network Dispatcher component, but again, you could very well choose to use the other components (as your requirements may dictate). We will use the term Load Balancer and Network Dispatcher interchangeably.

The controllers are interesting elements to discuss as they allow third-party switches like CSS and Alteon to be able to obtain metrics that allow them to do forwarding of requests taking into consideration z/OS characteristics. This makes possible a more complete portfolio of routing choices. For example, the CSS 11000 (Cisco's family of standalone, content-aware switches) has the ability to make routing decisions based on URLs, cookies, or host tags, but it does all its routing according to its own set of algorithms.

Edge Server's Network Dispatcher/Load Balancer is able to provide CSS with a matrix obtained through some of its advisors. One of the supported advisors is WLM 390. This feeding mechanism is done in a nonintrusive way so that it does not negatively influence the performance of the CSS's forwarding mechanism. Currently CSS has no other way of obtaining the z/OS-specific information that those advisors are able to provide.

The Network Dispatcher/Load Balancer creates edge-of-network systems that direct network traffic flow for all Internet servers, such as HTTP, FTP, HTTPS, and Telnet, reducing congestion and balancing the load on various other services and systems. Load Balancer provides site selection, workload management, session affinity, and transparent failover.

The Network Dispatcher/Load Balancer is installed between the Internet and the enterprise's backend servers, which can be content hosts or Caching Proxy machines. Load Balancer intercepts data requests from clients and forwards

each request to the server that is currently best able to fill the request. In other words, it balances the load of incoming requests among a defined set of machines that service the same type of requests.

The Network Dispatcher/Load Balancer can distribute requests to many types of servers, including WebSphere Application Servers and Caching Proxy machines. In one of our configurations we distribute requests to the Caching Proxy, and in the other configuration we distribute requests to the Web server.

As with each type of component (certain restrictions do apply), you can choose the type of TCP forwarding method you want your load balancer to use. A forwarding method with your load balancer answers the question: How will the TCP package be transmitted from the load balancer to your target servers and then how does it get back to the client.

We implemented the MAC forwarding method. Only the request from the client to the server (usually a small data packet) has to go through the Dispatcher. The response to the client can then be routed back directly to the client.

Using MAC forwarding, you can expect better performance and throughput than with any other method. But you can only use this method if the Network Dispatcher is on the same LAN segment as the server where the request has to be sent.

You have to define each Network Dispatcher separately. We started with the basic configuration of the primary Network Dispatcher, then added the high availability function to it. After this we defined the basic and high availability configuration of the backup Network Dispatcher.

Other methods of forwarding include something called IP tunnelling or Generic Routing Encapsulation (GRE) and Network Address Translation (NAT). Our tests didn't encompass these methods but we still tell you how to implement them on a Linux for zSeries platform.

The reason for doing this will become clear when you understand how forwarding takes place in 7.3.5, "Web server forwarding configuration" on page 164.

The Network Dispatcher distributes the load between multiple clustered servers (any type of TCP/IP server). For a highly available Web site, these are Web servers and Web Traffic Express servers.

As soon as a client request arrives, the load balancing machine uses sophisticated monitoring tools and forwards the request to the least loaded server. The Web server selected by the Network Dispatcher machine can respond directly to the client without any further involvement of the Network Dispatcher machine. Since there is no need for the server response to go back

through the same physical path, a separate high-bandwidth connection can be used.

You can use the Network Dispatcher component to balance the load on servers within a local area network or a wide area network using a number of weights and measurements that are set dynamically.

The Network Dispatcher/ load balancer function uses different forwarding methods:

- ▶ MAC forwarding
- ▶ NAT/NAPT forwarding

We describe these in detail in the following sections.

MAC forwarding

In this method, the Network Dispatcher receives the client's request packet and translates the destination Media Access Control (MAC) address of the packet from the Network Dispatcher MAC address to the load-balanced target server's MAC address. Since the IP packet header fields, specifically the client IP address, remain unchanged, the load balanced server returns the packet directly to the client.

This implementation is very fast and effective for performance, but requires that both the Network Dispatcher and load balanced target servers be on the same IP subnet. The Web servers can be located in the DMZ with Network Dispatcher, or on the same IP subnet as Network Dispatcher and not separated by a firewall.

Network Address Translation (NAT/NAPT) forwarding

Using this method, the Network Dispatcher can send the client's request not only to servers in the same IP subnet, but also to servers in a different IP subnet or IP network. This function translates the source IP address from the client's IP address into a Network Dispatcher address, called the *return address*, and translates the destination cluster IP address into a load-balanced server's IP address.

Network Dispatcher performs the reverse procedure for response packets from the servers. Each response to the client must go through the Network Dispatcher. This impacts the size of the Network Dispatcher server, and even the whole Web site performance.

The NAT function is of benefit any time you want to place the servers being load-balanced behind a firewall, or on a different IP subnet from the Network Dispatcher (for example, when locating the servers in a secure zone separate from Network Dispatcher).

Another instance may be when the servers being load-balanced are located at a separate and remote location, possibly kept separate for availability or disaster recovery.

Content-based forwarding

In some cases, not all the information needed to make load balancing decisions is found in the TCP/IP packet headers. Some information may be contained in the application-defined data that flows between the client and the back-end server. This implies the use of a proxy or proxy-like function that actually terminates the client's connection, reads the application data, and creates another connection to the chosen backend server.

In previous releases of IBM WebSphere Edge Server, the Content Based Routing component of the Load Balancer combined the load balancing, manager, and advisor functions of Network Dispatcher with the Caching Proxy server to permit load balancing based on the content of HTTP requests at the application level. This component of Network Dispatcher remains available in WebSphere Edge Server.

However, the Network Dispatcher component of WebSphere Edge Server V2 introduces a new, kernel-based, content-based routing (CBR) component that does not require the Caching Proxy.

Content-based forwarding is a means of deciding which target to select. Once a target has been selected, either content-based or by relative available capacity, one can then forward via four means:

- ▶ MAC forwarding
- ▶ NAT/NAPT
- ▶ Generic Routing Encapsulation (GRE)
- ▶ Proxying (establishing a second connection)

MAC and NAT/NAPT forwarding and GRE are normally used with capacity-based workload balancing. Proxy connections are used with content-based server selection.

A detailed description of all Network Dispatcher functions can be found in *Load Balancer Administration Guide Version 5*, GC09-4602.

3.3.2 Sysplex Distributor

The Sysplex Distributor is a state-of-the-art connection dispatching technology that is used among z/OS IP servers. In contrast to Network Dispatcher, the

dispatching entity in this solution is a z/OS system in a sysplex, and the target servers are exclusively z/OS systems in the same sysplex.

Sysplex Distributor extends the notion of automatic Virtual IP Address (VIPA) takeover to allow for load distribution among target servers in the sysplex. It combines routing technology used for the distribution of incoming connections with that of Dynamic VIPAs (DVIPA) to ensure high availability of a particular service in the sysplex.

The functionality of Sysplex Distributor is similar to that of Network Dispatcher in that one IP entity advertises ownership of some IP address by which a particular service is known. In this fashion, the single system image of Sysplex Distributor is also that of a special IP address.

However, in the case of Sysplex Distributor, this IP address (known as the cluster address in Network Dispatcher) is called a *distributed DVIPA*. Further, in Sysplex Distributor, the IP entity advertising the distributed VIPA and dispatching connections destined for it, is itself a system image within the sysplex, referred to as the *distributing stack*.

Like Network Dispatcher and DNS/WLM, Sysplex Distributor also makes use of Workload Manager (WLM) and its ability to determine server load. WLM informs the distributing stack of server loads so that the distributing stack may make the most intelligent decision regarding where to send incoming connection requests.

Additionally, Sysplex Distributor has the ability to specify certain policies in the Policy Agent so that it may use QoS information from target stacks in addition to the WLM server load. Further, these policies can specify which target stacks are candidates for clients in particular subnetworks.

As with Network Dispatcher, connection requests are directed to the distributed stack of Sysplex Distributor. The stack selects which target server is the best candidate to receive an individual request, and routes the request to it. It maintains state so that it can forward data packets associated with this connection to the correct stack. Data sent from servers in the sysplex need not travel through the distributing stack, but can travel directly to the destination address.

The connection routing technology of Sysplex Distributor allows a VIPA to move nondisruptively to another stack. In the past, a dynamic VIPA was allowed to be active only on one single stack in the sysplex at a time. This led to potential disruptions in service when connections existed on one stack, yet the intent was to move the VIPA to another stack.

With the connection routing technology, the movement of VIPAs can now occur without disrupting existing connections on the original VIPA owning stack. Only

the new stack advertises ownership of the VIPA to the outside network, and it routes connection data to the original owning stack for connections that terminate there, as long as those connections last.

It should be noted that Sysplex Distributor does not support affinity to a specific host. The choice is selected according to WLM metrics, policies, and configurations and could possibly be a different target than in the last request. This is important when considering Web-based applications that need to keep state between requests. A WebSphere topology configured with Sysplex Distributor will not be able to enforce session affinity.

This combination works well with stateless applications. For applications that need to keep state, the WebSphere plug-in should be the one in charge of dispatching directly to the WebSphere servers.

IIOp workflow in a Parallel Sysplex

As we may have Java clients communicating IIOp instead of Web-browser clients using HTTP, this is a special scenario that needs attention.

Usually clients use browsers to communicate using HTTP over TCP/IP and coming through multiple layers (Edge Servers, HTTP servers, etc.) till getting to Sysplex Distributor. This book is built on that scenario, as you can see in the remaining chapters.

However, there are situations when Java clients are built using IIOp communication. Figure 3-3 shows the workflow for an IIOp request in a sysplex-enabled Network Deployment configuration. DVIPA addresses are used for the Node Agents and Location Daemons to take advantage of Sysplex Distributor workload balancing. Clusters are used to allow the workload to be balanced horizontally across LPARs in the sysplex.

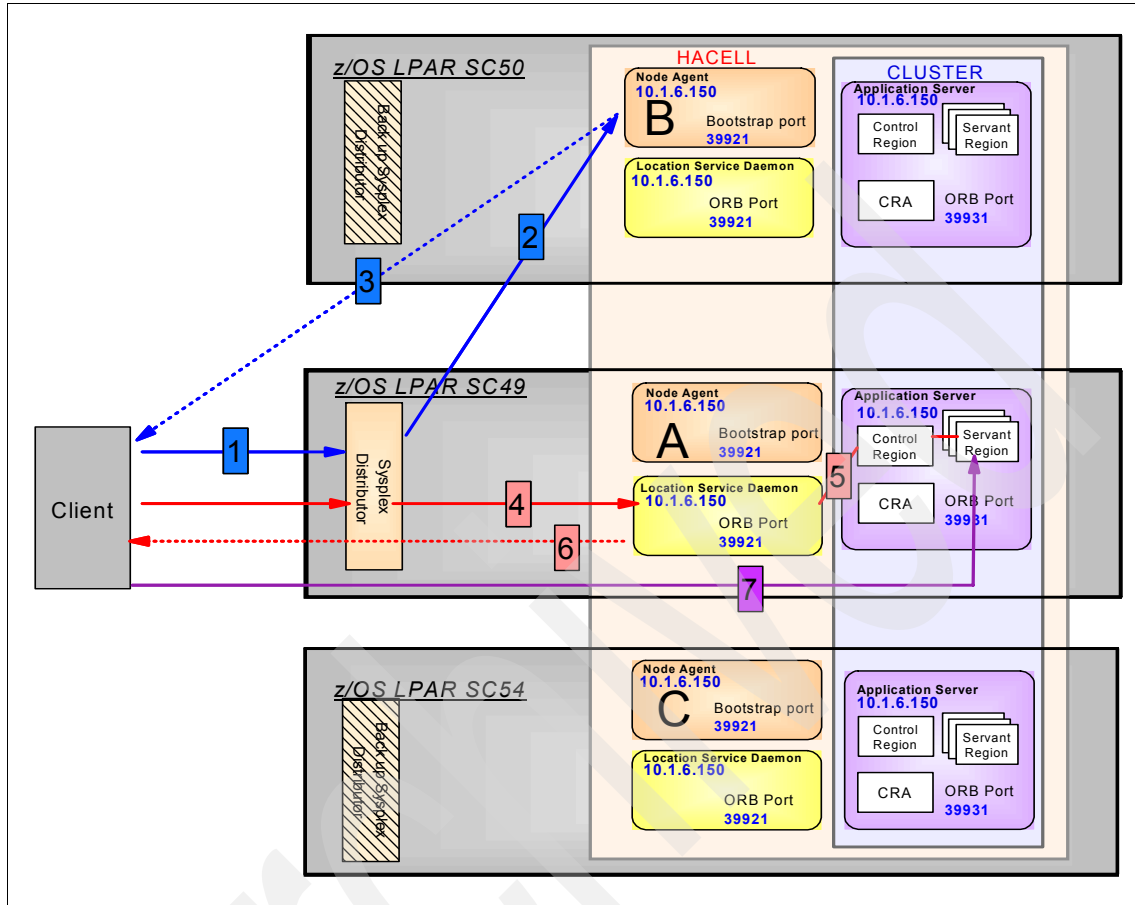


Figure 3-3 IIOP flow in a sysplex

The numbers in the graphic correspond to the following:

1. Name lookup on WebSphere z/OS is carried out through bootstrapping to the Node Agents, which all listen on a single DVIPA address.
2. The Sysplex Distributor balances the name lookup requests across all Node Agents in the sysplex.
3. The Node Agent returns an indirect IOR for the EJB object or Home looked up. The indirect IOR contains the DVIPA address of the Location Daemons on z/OS.
4. Once used by the client, the indirect IOR goes through the Sysplex Distributor, which “workload manages” the request to a suitable daemon.

5. The daemon uses Workload Manager to determine which Application Server should handle the request.
6. The daemon returns a direct IOR, which contains the forwarded location of the Server Cluster member that will handle the request.
7. The client ORB uses this direct IOR to send the request directly to the specific cluster member.

3.3.3 Sysplex Distributor and MultiNode Load Balancing (MNLB)

MultiNode Load Balancing (MNLB) is Cisco's connection dispatching technology. It consists of software running in the Sysplex Distributor component, version z/OS 1.2 or later, and specific Cisco routers or switches that have been enabled for such support.

Where the dispatching entity is in the PC or workstation with Network Dispatcher and in the sysplex with Sysplex Distributor, the dispatching entity is located in the router or switch with MNLB.

The MNLB architecture is comprised of the following four components:

- ▶ Service Manager
- ▶ Forwarding Agent
- ▶ Workload Agent
- ▶ Backup Service Manager

Figure 3-4 on page 67 shows the location of the MNLB components.

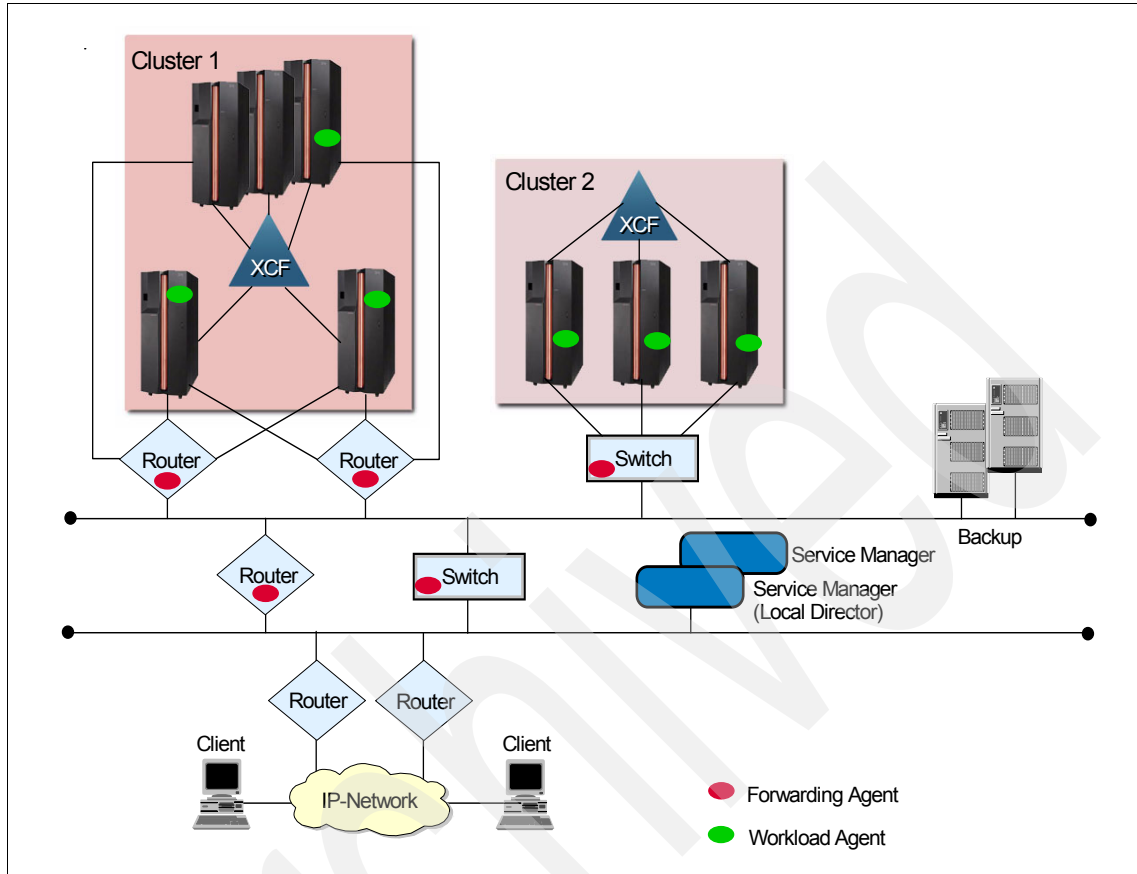


Figure 3-4 MNLB components

A *Service Manager*, such as the Cisco Local Director, is responsible for the distribution of connection requests. The Service Manager makes the decision as to which target server will receive a connection. This is done using information about application availability, server processor capacity, and a load-balancing algorithm such as round robin, least used connections, or based on information received through the Dynamic Feedback Protocol (DFP). This protocol, for example, carries information from the Workload Manager (WLM).

A *Workload Agent* provides the information to the Service Manager that needs to calculate an optimal load-balancing result for the server selection. The Workload Agent is software that runs on server platforms or machines that manage server farms or clustered server environments. The Cisco Workload Agent for OS/390 uses WLM data. It converts this data into the common DFP protocol before it is

sent to the Service Manager. The Cisco Workload Agent for OS/390 optimizes load balancing in an IBM sysplex environment.

A *Forwarding Agent* is used as a packet redirector that forwards packets based on the Service Manager's instructions. The Forwarding Agent is software running in Cisco routers or switch modules. There can be multiple Forwarding Agents for the same distributed service (application). A *Backup Service Manager* is responsible for providing connection establishment when the primary Service Manager fails.

In an MNLB approach, Sysplex Distributor executes the role of the Service Manager. The idea behind this union is to use the Cisco routers to do what they do very well, route requests. The Sysplex Distributor is integrated with the z/OS subsystems, enabling it to intelligently advise the routers (Forwarding Agents) on the most appropriate dispatching targets.

HTTP sessions

An important area to consider when designing a J2EE infrastructure is how to manage session state. Session state is a working set of information maintained across multiple user requests, sometimes referred to as *conversational state*. The terms conversational state and session state can be used interchangeably, although session state will be used throughout this chapter. Here we introduce the various mechanisms for maintaining session state and discuss the considerations for scalability, availability, and performance.

4.1 Overview

In many Web applications, users dynamically collect data as they move through the site based on a series of selections on pages they visit. Where the user goes next, and what the application displays as the user's next page (or next choice) may depend on what the user has chosen previously from the site. For example, if the user clicks the checkout button on our site, the next page must contain the user's shopping selections. Session state is a working set of information that represents a conversation. A user dynamically collects this working set of data as they move through a series of requests or pages. It provides the link between the requests that build up the overall conversation.

In order to do this, a Web application needs a mechanism to hold the user's state information over a period of time. However, HTTP alone does not recognize or maintain a user's state. HTTP treats each user request as a discrete, independent interaction.

The Java servlet specification provides a mechanism for servlet applications to maintain a user's state information. This mechanism, known as a *session*, addresses some of the problems of more traditional strategies such as a pure cookie solution. It allows a Web application developer to maintain all user state information at the host, while passing minimal information back to the user via cookies.

Where the state is kept and how it is maintained can have a large impact on the infrastructure for scalability, availability, recoverability, performance, workload management, and security, and unless you use the right tools for the job you could find that you are seriously compromised in one of these areas.

4.2 Maintaining session state

Applications can be stateful and stateless and this should be clearly differentiated from the maintenance of session state. A stateless application means that each request that passes through the application has no bearing on the previous or next request. A stateful application is an application where one request can be impacted by the previous or next request or needs to keep information about the ongoing requests that pass through.

To provide the optimally performing and scalable solution, the application should be stateless, because this allows for very effective workload management across instances of the components in the application. The use of a stateless application does not preclude the use of session state, it is just that all the state for the application is maintained in the session, and none in the application components

themselves. This does require the Web components to acquire, pass the state, and update the state during the application conversation with the user.

Not all applications require session state to be maintained. The choice here should be based on whether the application is conversational in behavior or non-conversational. For a non-conversational application there should be no need to keep state between requests because there is no conversational data built up. During the early stages of application design this issue should be considered, and where possible the stateless approach should be taken. Interaction diagrams, such as sequence diagrams, are very good for showing up where conversational state is required.

From the above statement we can see that we have contradictory needs: One to maintain a stateless application to achieve best workload management, and the other to handle a conversation that requires state to be managed.

In storing state, one should attempt to identify those elements that make up the state of the application or conversation. Then establish whether this is only a conversation state with a minimum set of information or is actually the application that requires state, in which case stateful components in the application are required.

4.2.1 Client tier

HTTP thin clients running in a browser, or fat client applications can represent this tier. By saving state in this tier you are relieving the responsibility from the server side of the infrastructure, thus allowing optimal scalability of server side components. The memory requirements for the session state are distributed over all the clients on different machines.

Using the client to maintain session state is only really practical when the amount of data is minimal. In fact, for stateful applications there will always be a requirement to maintain at least some state on the client so that the user can retrieve their correct session state further down the infrastructure.

For example, if using HTTP sessions in the presentation layer, a cookie could be stored containing the session ID of the HTTP session so that the correct session object can be retrieved with each request.

With large amounts of session data being transferred back to the client, the cost of the network traffic will increase as the application scales, thus reducing overall performance and possibly creating bottlenecks in throughput. If the session data is sensitive, then it must be encrypted before being transmitted to and from the client, which would entail further CPU cost on both the client and the server.

A further processing cost would be that of the transformation of the data to and from a format that the client can maintain—typically text-only. Such parsing could be costly for complicated or large data structures. For the clients then these costs would be dispersed over each client machine, but on the server side these costs would be met centrally by the infrastructure, adding weight to the application's CPU, memory, and I/O requirements.

One final disadvantage of using clients to save session state is the lack of centralized control over the quality of this mechanism. The session state will be at the mercy of each client's environment. If the client crashes, there is no guarantee that the state is recoverable.

Mechanisms for maintaining session state in the client tier

These mechanisms are only valid to clients that support the HTTP protocol. Fat clients will have their own proprietary programmatic means of maintaining client side state.

- ▶ **Cookies**

A cookie is a text-only string that gets entered into the memory of your browser and can be stored in the client's file system (if the cookie settings allow). It is created either on the client side under the browser's processing (for example, cookies can be manipulated with Javascript embedded into an HTML page which is then executed under the browser), or on the server side from servlets/JSPs using the Cookie class.

Cookies can either be session scoped or persistent. That is, they can be set to expire after a certain time, when a session is closed, or persisted by the browser until cleaned out, based on the browser/client policy for cookie management. The persistent use of cookies can be very useful if you want certain session data to be carried forward to future sessions.

- ▶ **HTML hidden fields**

By using HTML hidden fields, text-only data can be passed back to the client in the body of the HTML itself. This data will not be displayed but it will be available to client-side scripting languages (such as Javascript) and also be in a <form> element so that it could be passed back to the server in the next request.

4.2.2 Web tier - HTTP sessions

Storing session state on the Web tier typically means using HTTP sessions maintained by the Web container. HTTP is a stateless protocol, treating each user request as a discrete and independent transaction. The Java servlet specification provides a mechanism for servlet applications to maintain a user's state information. This mechanism—known as an HTTP session—addresses

some of the problems of client-side strategies, such as a pure cookie solution. It allows a Web application developer to maintain all user state information on the host, while passing minimal session tracking information back to the client. Storing session state in the presentation tier reduces the amount of data passed over the network.

WebSphere has a Session Manager that maintains all aspects of HTTP session state. The Session Manager provides a rich set of features that can be used to provide functionality above that of the servlet specification, such as persisting sessions, session replication, session caching, and expanding session scope. Each of these will be talked about in detail in the following sections.

Session tracking

Once an HTTP session has been established, it is identified by a session ID. Subsequent requests from the user must be associated with that same session ID. The HTTP protocol is stateless, so an additional mechanism must be used to maintain the session tracking.

The Session Manager module, which is part of each Web container, is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request.

The Servlet 2.3 API Specification defines three mechanisms for HTTP session tracking. You can set a combination of methods in the Web container's session management properties. The following order of preference is used: SSL session, cookies, URL rewriting.

SSL sessions

This method uses the stateful HTTPS protocol. The Session Manager maintains the session ID on the established SSL connection so no coding is required by the developer.

Since this mechanism maintains the affinity from the client to the inbound port of the Web server and the Web server then passes the request to any number of application servers through the WebSphere plug-in, the use of this affinity mechanism for session affinity only works where there is a one-to-one relationship between the Web server end point for the client and a single application server instance behind this Web server.

In the case where multiple application server instances exist, cookies must be used.

Cookies

This is a simple mechanism to implement and is the most commonly used method to track sessions. No coding is required to take advantage of this tracking mechanism, although this method can only be used if the client's browser accepts cookies.

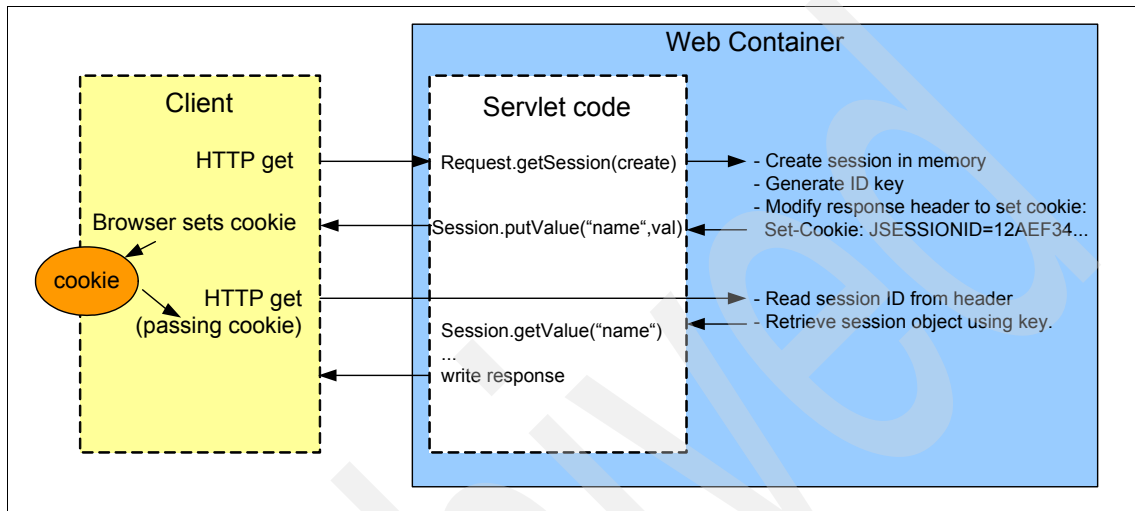


Figure 4-1 HTTP session tracking using session cookies

In our configuration, we implemented HTTP sessions based on cookies. The cookie is created by WebSphere on behalf of an application request and sent to the client. The client (browser) stores the cookie in memory. A session cookie is not stored in a file, and will disappear when the browser terminates. See Figure 4-1 for cookie processing.

URL rewriting

With URL rewriting, the Session Manager includes the session ID in the HTTP response when the requested servlet or JSP issues the `encodeURL()` method on the response object. This method is not suitable if you have requests for static HTML pages because the session ID cannot be encoded and is lost.

URL rewriting can be used in combination with cookies. It is particularly useful in situations where a client cannot use cookies, for example, if the client is a Web-enabled cell phone.

4.2.3 Session affinity

The default behavior for the Session Manager is to hold HTTP sessions in memory. If you choose to use session affinity, then the workload balancing

methods have to honor this affinity to a specific server clone and servant region. This affinity is tracked using a cloneID token that is passed along with the request when using session cookies or URL rewriting.

In a non-z/OS environment, it is the WebSphere plug-in that maintains the affinity to the application server instance.

Note: in a z/OS WebSphere V6 environment, session affinity is associated with a particular servant. If the servant with specific session affinity should fail, any of the other servants can retrieve the HTTP session data stored in the controller and establish a new affinity.

To evenly distribute work to different servants, WLM features a function called “even distribution” of HTTP requests. For applications with no HTTP sessions or short HTTP session requests, you can use the default workload distribution strategy, which picks a hot servant to handle the new request for better performance. A hot servant has just had a recent request dispatched to it and has threads available as well as resources in memory.

For applications with many HTTP sessions and long HTTP sessions, the WLM “even distribution” of HTTP requests function supports distribution of incoming HTTP requests without servant affinity in a round-robin manner across the servants. When a new HTTP request without affinity arrives on a work queue, WLM dispatches the request to a servant with an available worker thread and the smallest number of affinities.

Considerations when deciding on session affinity

Session affinity impacts workload balancing since the requests must be processed by the servant region that contains the in-memory session data. With affinity turned on in the Web server WebSphere plug-in, only new requests and requests which do not have a session associated with them are eligible for workload management.

On the plus side, maintaining session state in memory can provide performance gains since the in-memory session acts as a cache and provides quick access to the data without the need to work with serialized objects.

Session persistence mechanisms that are described in the following sections require serialization of the session objects. Serialization is a very costly process, especially when dealing with complex objects.

Furthermore, DB2 persisted session data is kept in a global buffer pool (GBP). Each DB2 subsystem in the data sharing group keeps a local bufferpool in storage. Unless data in this buffer pool is changed—and therefore marked

invalid—the information is read from this storage. By maintaining some degree of affinity, these GBPs can be used most efficiently to provide quicker access to session data than going to the database itself.

You should also consider the size of the session data in order to capacity plan for the memory footprint. Consider the size of the average sessions, the estimated lifetime of the sessions, and then calculate over this lifetime how many concurrent clients could have established sessions. If the memory footprint is too large, then a combination of session affinity and session persistence may provide the answer.

In this scenario the in-memory cache is used, but it can overflow to the database when the number of sessions reaches the max in-memory session count.

4.2.4 Session recovery and failover

Besides storing session objects in memory, there is also the option to enable mechanisms that will allow the session to be recovered after an application server failure. The user requests can then be processed on another application server in the cluster with the session information having been retrieved from the failover mechanism.

As well as providing recovery, these mechanisms also allow session affinity to be turned off in the Web server plug-in. In such a situation the session can be retrieved from any application server in the same cluster. Affinity should be considered carefully.

Performance data from the various failover mechanisms and the use of a non-affinity based mechanism vary dramatically with different infrastructures and should be carefully considered here.

A session recovery mechanism should be enabled when:

- ▶ The user's session data must be recovered by another server instance after an instance in a server group fails or is shut down.
- ▶ The user's session data is too valuable to lose through unexpected failure at the WebSphere node.
- ▶ You do not want to use affinity-based session support in the Web server plug-in.
- ▶ The administrator desires better control of the session cache memory footprint. By sending cache overflow to a persistent session database, the administrator controls the number of sessions allowed in memory at any given time.

WebSphere provides two mechanisms for session recovery: session persistence and session replication.

Recovery vs. performance

Saving session state does not completely guarantee the state in the event of a failover. If the failure occurs while the state is being written, then this would be lost. This is a small window for loss of session data. Because writing the data can be costly—due to serialization and database access—the amount of delayed writing can improve the performance and response time dramatically.

By only writing updated data, and by selecting to write out either manually (programmatically triggered using `IBMSession.synch()`) or delayed, then the cost of persisting the session is reduced.

This introduces a greater risk during failover, but this risk should be mitigated to what is acceptable in your infrastructure. The loss of a certain period of session data may be quite tolerable depending on the application and what the session is storing.

4.2.5 Database session persistence

The WebSphere Application Server for z/OS V6 has two ways of providing session persistence.

The first and more robust way to persist session objects is using a database. If a server fails, another server may retrieve the session data, thus allowing the client request to continue to be processed. As the session data is saved in a database, this is the only viable solution possible in case of a total WebSphere outage, where all servers need to be restarted. On the other hand, if the database is not available either, the session data is also unavailable.

Session persistence is the most common method of providing session recovery in WebSphere. Sessions are serialized and persisted to a database where any application server in the same cluster can retrieve them.

Since all information stored in a persistent session database must be serialized, all the objects held by a session must implement *java.io.Serializable*. In general, consider making all objects held by a session serialized, even if immediate plans do not call for the use of persistent session management.

Why? So that if the Web site grows, and persistent session management becomes necessary, the transition between local and persistent management occurs transparently to the application if the sessions hold only serialized objects. If that is not the case, a switch to persistent session management requires coding changes to make the session contents serialized.

You need to decide whether to use a single-row or multi-row database schema to store your HTTP session data. Making the right decision will have an impact on

performance. To help make the decision you should try both options on a representative application to monitor and access the performance results.

Single-row schema

All session data is stored in a single row of the database. There is a limit of 2 MB for each HTTP session stored in this way. For smaller and fewer session objects this method is efficient since you can access all values in just one record.

Multi-row schema

Each piece of data is stored in a separate row of the database. Apart from the size of your database, there is no imposed limit on the size of the HTTP session that can be stored. Each object stored in the session is written and accessed individually to a separate row. This provides a performance benefit when storing many objects because you do not have to access and serialize the whole session data at once. However, if you are using relatively small and few session objects, then accessing multi-row schema may prove to be a performance overhead.

4.2.6 Memory-to-memory session replication

The other mechanism provided by WebSphere Application for z/OS V6 for session management is called memory-to-memory session replication. The sessions can be replicated to one or more WebSphere Application Server instances.

The HTTP plug-in sends a new request to a server instance. A session is created and session affinity is established. The session is now replicated to other server instances. If the original server instance fails, the HTTP plug-in will route the user's request to another server instance in the cluster.

The session manager of the new server instance either retrieves the session from a server instance that has the backup copy of the session, or it retrieves the session from its own backup copy table. The new server now becomes the owner of the session and affinity is now maintained to this server.

Note: In the WebSphere Application Server for z/OS V6, servers that are enabled for HTTP session memory-to-memory replication store the HTTP session data in the controller and replicate it to other WebSphere Application Servers.

Server modes

There are three possible modes a server instance can run in:

- ▶ Server mode - Only store backup copies of other WebSphere Application Server sessions and not send out copies of any session created in that particular server.
- ▶ Client mode - Only broadcast or send out copies of the sessions it owns and not receive backup copies of sessions from other servers.
- ▶ Both mode (default mode) - Simultaneously broadcast or send out copies of the sessions it owns and act as a backup table for sessions owned by other WebSphere Application Server instances.

Topologies

The following are the two primary topologies for memory-to-memory replication:

- ▶ Peer-to-peer replication
- ▶ Client/server replication

In a cluster, by default, sessions are replicated in all the servers in the cluster that are connected to the same replicator domain. This replication can be redundant if a large number of servers exist in a cluster. The session management facility has an option to partition the servers into groups when storing sessions.

Note: The cost to the JVM heap size should be determined when deciding which replication method to use. Peer-to-peer provides the simplest method to configure and scale, but it also comes with the highest memory cost.

Peer-to-peer replication

The basic peer-to-peer (both client and server function, or both modes) topology is the default configuration and has one replica.

Note: You must upgrade all WebSphere Application Server plug-in instances that front the Application Server cluster to Version 6.0 to ensure session affinity when using the peer-to-peer mode.

In this basic peer-to-peer topology, each server Java Virtual Machine (JVM):

- ▶ Hosts the Web application leveraging the HTTP session.
- ▶ Sends out changes to the HTTP session that it owns.
- ▶ Receives backup copies of the HTTP session from all of the other servers in the cluster.

Replication Strategy

You can define a replication domain and specify how many replicas are in the domain. Figure 4-2 shows two replication configurations. The one on the left has a single replica. The one on the right has four replicas.

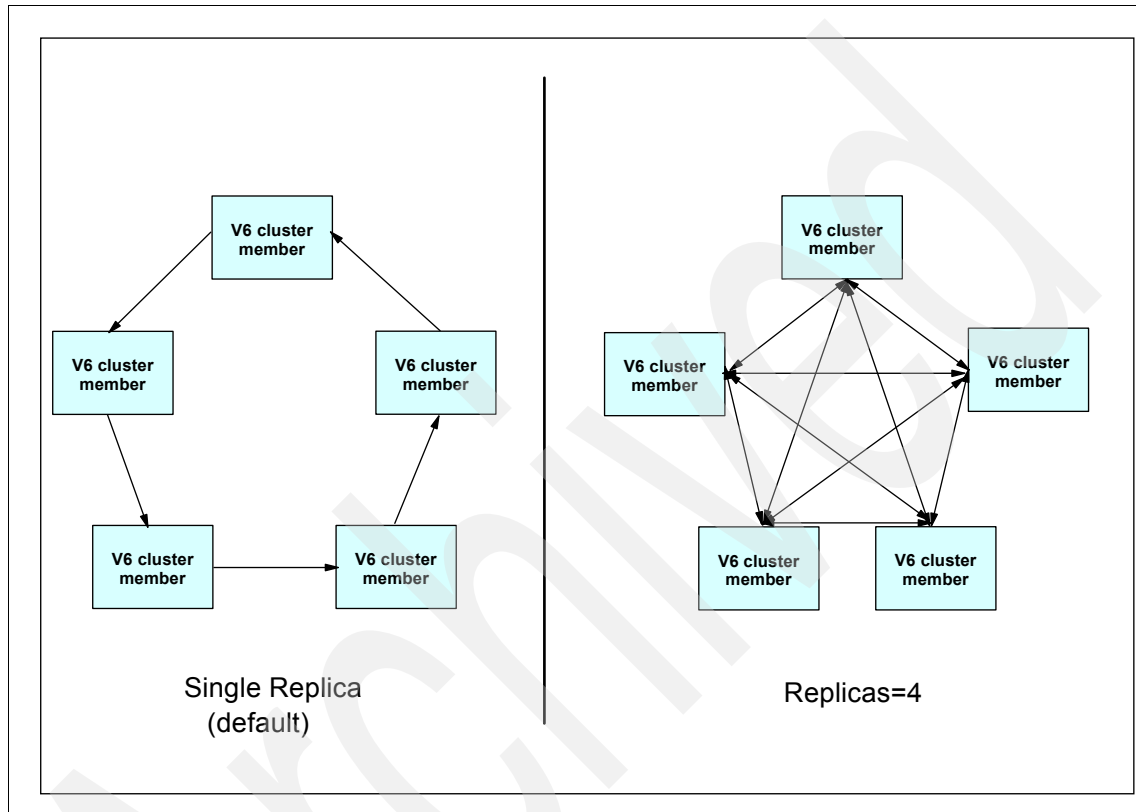


Figure 4-2 Replicas in a replication domain

Using the whole domain as replicas for every application server is the most redundant because everyone replicates to everyone. Thus in any failure recovery scenario, the server routed to already has a copy of the session.

But as you add servers, more overhead (both CPU and memory) is needed to deal with replication. You need to balance this resource consumption with your failover “comfort level”.

Specifying peer-to-peer replication

Figure 4-3 shows the specification for replication domain from WebSphere administrative console.

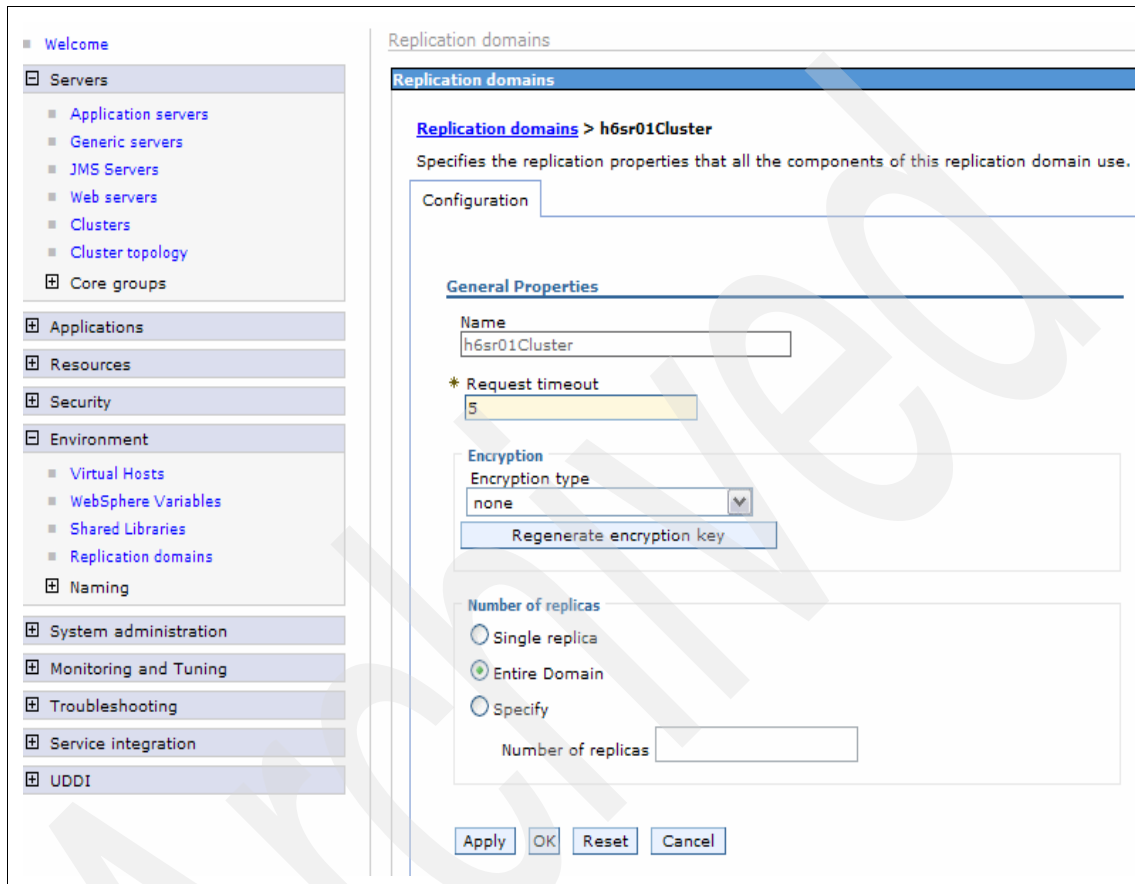


Figure 4-3 Setting of number of replicas

In WebSphere Application Server V6, a new feature called *session hot failover* has been added to the peer-to-peer mode.

In a clustered environment, session affinity in the WebSphere Application Server plug-in routes the requests for a given session to the same server. If the current owner server instance of the session fails, then the WebSphere Application Server plug-in routes the requests to another appropriate server in the cluster.

For a cluster configured to run in the peer-to-peer mode, this feature causes the plug-in to failover to a server that already contains the backup copy of the

session, therefore avoiding the overhead of session retrieval from another server containing the backup.

Client/server replication

Client/server mode topology is still configurable in Websphere Application Server V6.

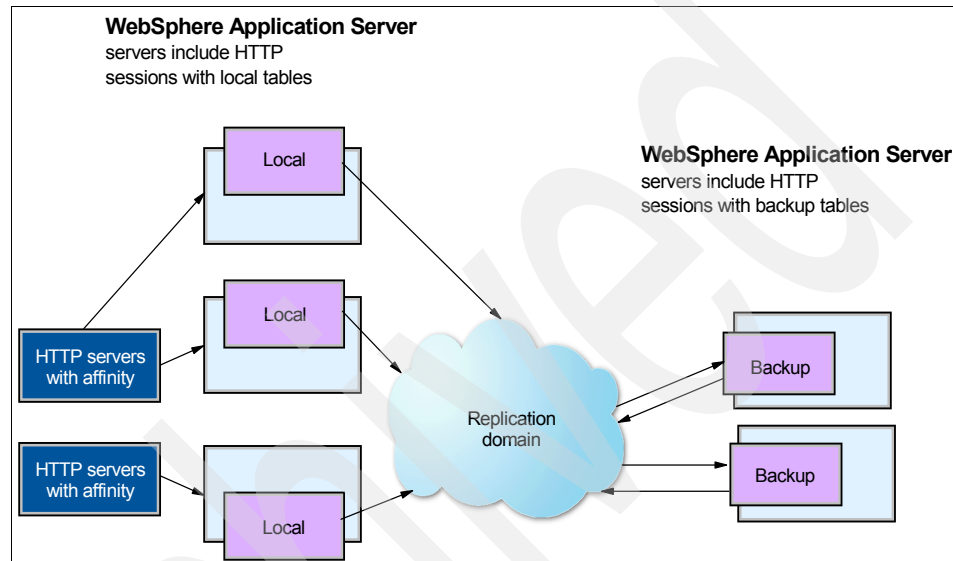


Figure 4-4 Client/server replication topology

In this case there is a tier of application servers that host Web applications using an HTTP session, and these sessions are replicated out as they are created and updated. There is a second tier of servers without a Web application installed, where the Session Manager receives updates from the replication clients. The replicators facilitating the transfer of data reside with the replication servers.

Benefits of the client/server with remote replicator configuration include:

- ▶ Isolation (for failure recovery)

In this case we are isolating the handling of backup data from local data; aside from isolating the moving parts in case of a catastrophic failure in one of them, you again free up memory and processing in the servers processing the Web application, much the same as the isolating of the replicators as shown in the topology for peer-to-peer function with remote replicators.

- ▶ Isolation (for stopping and starting)

You can recycle a backup without affecting the servers running the application (when there are two or more backups, failure recovery is

possible), and conversely recycle an application JVM without potentially losing that backup data for someone.

- ▶ Consolidation

There is most likely no need to have a one-to-one correspondence between servers handling backups and those processing the applications; hence, you are again reducing the number of places to which you transfer the data.

- ▶ Disparate hardware

While you run your Web applications on cheaper hardware, you may have one or two more powerful computers in the backend of your enterprise that have the capacity to run a couple of Session Managers in replication server mode, allowing you to free up your cheaper Web application hardware to process the Web application.

- ▶ Timing consideration

Start the backup application servers first to avoid unexpected timing windows. The clients attempt to replicate information and HTTP sessions to the backup servers as soon as they come up. As a result, HTTP sessions that are created prior to the time at which the servers come up might not replicate successfully.

Specifying client/server replication

You can define replicators on both the replication client and the server JVMs. However, having replicators on both clients and servers is redundant. In the client/server topologies, defining replicators only on the backup replication server JVMs is recommended.

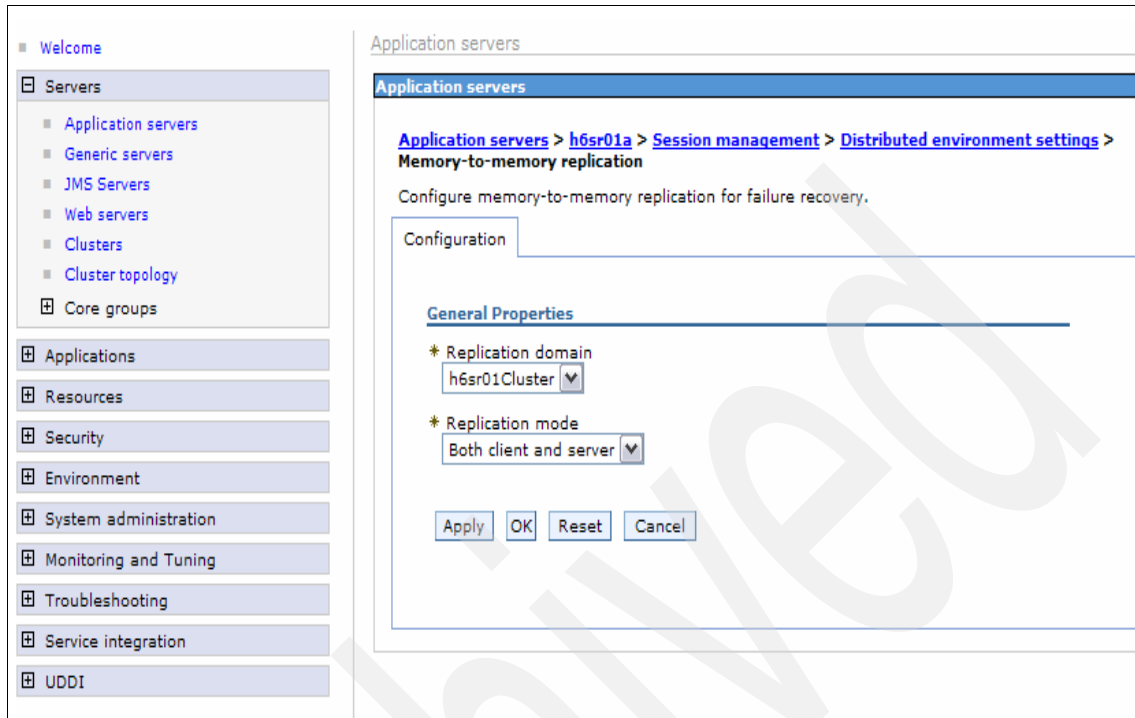


Figure 4-5 Client/server replication mode setting

It is good to spread the replication clients across the replicators as equally as possible (by default they select the first replicator in the domain), because both replication servers are doing work and not acting as a hot standby.

Timing considerations: Start the replication servers first to avoid unexpected timing windows. The clients attempt to reconnect to the replication domain if you start the replication clients before the replication servers, even if the initial connection cannot be completed. However, if servers running the application come up, and requests to the applications occur before the replicators on the backup servers have finished coming up, some expected client replication might not occur.

4.3 Data flow in HTTP sessions with affinity

The next sections describe the data flow between the client and the server via the workload distribution mechanisms we used in our project:

- ▶ Using session persistence with an internal replication domain

- ▶ Using session persistence with a DB2 database

Both scenarios exploited session affinity using the IHS plug-in.

4.3.1 Session persistence using an internal replication domain

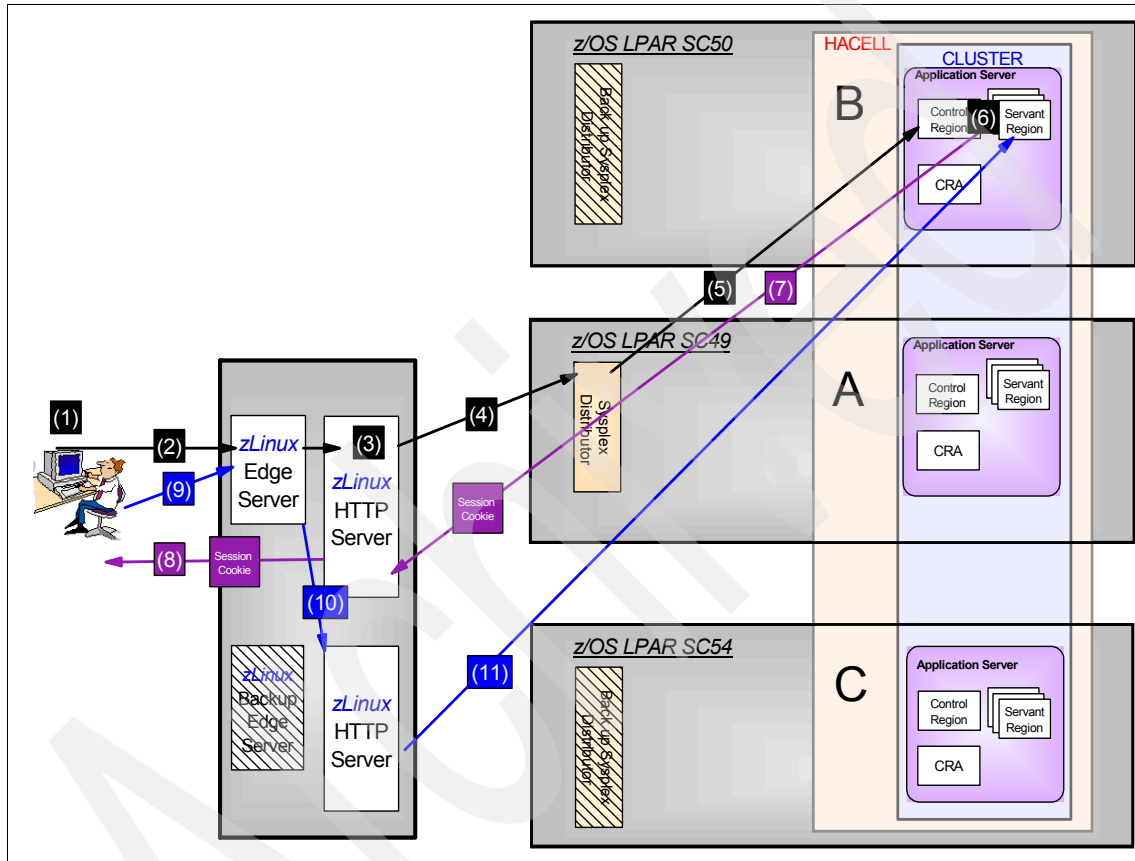


Figure 4-6 Request flow with session affinity using internal replication

The following notes refer to the highlighted sections of Figure 4-6.

(1) The end user enters an initial request to the browser; the request has no session affinity.

(2) The request resolves and is sent to the active Edge Server (ES) running the Network Dispatcher.

(3) Both HTTP Servers are defined in the ES. The ES sends the request to one of the IBM HTTP Servers (IHSs). Both IHSs are configured with the same plug-in configuration file.

(4) The plug-in checks for possible session affinity by scanning the request for session cookies. The first request has no affinity. The plug-in recognizes it and routes it to the DVIPA address of the Sysplex Distributor (SD).

Note: If you do not use Sysplex Distributor (or if your plug-in does not support a cluster address), the requests without session affinity will be sent to one of the available systems.

A typical plug-in config xml file would not have the cluster entry, and you may customize the plug-in behavior regarding how it will send the requests forward to receiving application servers (round-robin, weighted, etc.).

Consult the latest version of the Infocenter for more details on how to change the plug-in config xml file.

The address of the SD can be specified in the plug-in config xml file. See Example 4-1. All requests are routed to the SD, which distributes the requests without session affinity.

Example 4-1 Portion of the plugin-cfg.xml file

```
<ClusterAddress name="haplex">
    <Transport hostname="HAPLEX1.ITSO.IBM.COM"
port="39938" protocol="http"/>
</ClusterAddress>
<Server CloneID="h6sr01aSC49" Name="h6nodea_h6sr01a">
    <Transport Hostname="10.1.6.20" Port="39918"
Protocol="http"/>
</Server>
<Server CloneID="h6sr01bSC50" Name="h6nodea_h6sr01b">
    <Transport Hostname="10.1.6.24" Port="39918"
Protocol="http"/>
</Server>
<Server CloneID="h6sr01cSC54" Name="h6nodea_h6sr01c">
    <Transport Hostname="10.1.6.22" Port="39918"
Protocol="http"/>
</Server>
```

The Sysplex Distributor (SD) has the definitions of the port and IP addresses on which the application servers listen.

(5) SD sends the request to the application server instance on the system with the best available resources. Sysplex Distributor gets this information from WLM. In Figure 4-6, the request is sent to system SC50.

(6) The HTTP listener located in the Controller region (CR) of the application server instance receives the request. Since there is no session affinity, the CR puts the request in the WLM queue for that application server and the next available servant region (SR) picks it up.

Now the request has entered the Web container of the J2EE server. If the application starts a session, it creates a session object and an associated session ID with that object. The container adds a session cookie to the HTTP data stream in the response to the client.

To add a session cookie, the WebSphere configuration must be set up to support session affinity using cookies. To enable session cookie support at a server level using the WebSphere admin console, go to **Servers** → **Application servers** → **server_name** → **Web container settings** → **Session management**, and check **Enable cookies**.

At this point, the session object is replicated via DRS memory-to-memory replication or persisted to DB2, making it available to other members of the cluster, for failover purposes.

The name of the session cookie is JSESSIONID; Example 4-2 shows an example of such a cookie. The first 4 digit are the cache ID, followed by the session ID.

Example 4-2 Example of the JSESSIONID cookie

```
JSESSIONID:0001JHvfgaUkD1AqulF19PkQ5Ym:h6sr01aSC49
```

Example 4-3 lists the structure of the JSESSIONID cookie.

Example 4-3 Structure of the JSESSIONID cookie

```
JSESSIONID:<cache validity status field><SessionID>:<CloneID>
```

The JSESSIONID cookie carries the cache validity status field (4 digits), session ID, the information about the right server region, and the CloneID. The CloneID is assigned to an application server during its creation. With this information, the correct application server control region (clone) can be found.

We noticed that when DRS is disabled, the CloneID is set to the clone ID when the cluster is created. If DRS is enabled, the CloneID in the JSESSIONID changes every time the server is recycled.

To get a fixed CloneID in the JSESSIONID, you can set a specific HttpSessionCloneId for each server. For example, in Figure 4-7 on page 88 we set the HttpSessionCloneId to h6sr01asc49 such that the CloneID portion of the JSESSIONID in Example 4-2 on page 87 is fixed to h6sr01asc49.

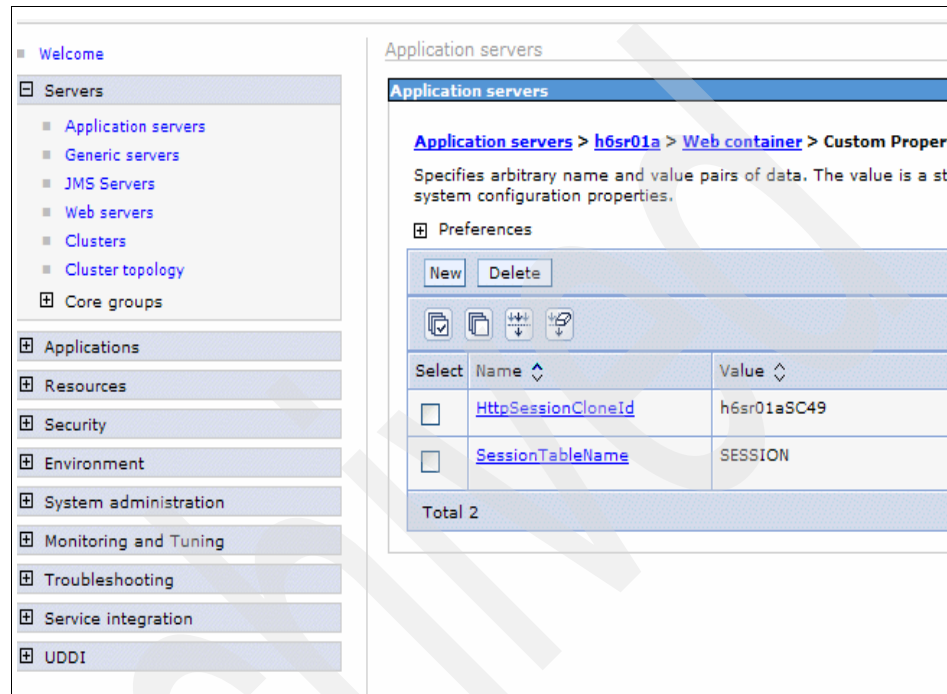


Figure 4-7 Set HttpSessionCloneId

(7) The response with the cookie in the HTTP header is sent back to the Web server.

(8) The response is sent directly to the client without going through the ES. This is possible because we use the MAC forwarding method in the ES.

The browser stores the cookie in its memory. (Note: Session cookies are never stored on a disk.) The cookie stays there as long as the browser is not closed, or until it is deleted (because of expiration) by the server.

(9) The cookie is appended to the HTTP request at the next request to the server.

(10) The ES does not have to consider session affinity, in our configuration. Therefore, there is no cookie check and the request could be sent to a different HTTP server.

(11) The plug-in scans the HTTP header of the request and finds the JSESSIONID cookie with a CloneID (h6sr01asc49).

It compares this CloneID with its definition in the plug-in configuration (plugin-cfg.xml) file. If there is a match, the request is sent to the IP address and port of the matched application server instance (in our example, to the application server instance H6SR01A on system SC49). This is done directly, without going to the Sysplex Distributor.

If HTTP session failover happens, WebSphere appends the HttpSessionCloneID of the failed over server to the JSESSIONID. See Example 4-4. The HTTP Server plug-in uses the far-right CloneID, in this example h6sr01cSC54 for server instance H6SR01C on SC54, as indication to where the HTTP session is located.

Example 4-4 JSESSIONID after HTTP session failover

JSESSIONID:0002JHvfgaUkD1AqulF19PkQ5Ym:h6sr01aSC49:h6sr01cSC54

The HTTP Transport Handler in the control region of the server instance WHSR01A is looking for the WlmservletID in the cookie. It finds the WLM token belonging to a specific Server Region. The request is put into the WLM queue, but it can only be picked up by that specific Server Region.

The flow between client and server continues until one of the following occurs:

- ▶ The session is closed by the application (that is, until the session object with the associated session ID is deleted).
- ▶ The user was inactive for a long time on that session. The session invalidation timer (defined in the webcontainer.conf file) deletes a session if there is no activity during the time interval defined by that parameter.

If for any reason the SessionID becomes invalid in the server, a client still sends the cookie stored in its memory to the server at its next request. The session affinity is still honored by the plug-in, and the request ends up in the server region where the last session was established. In this case, there is no session object with a SessionID stored in the received cookie.

WebSphere creates a new session object with a new SessionID and sends the new cookies back to the client. The CloneID is still the same. Because of the old, invalid cookie, the request ends up in the same servant region. The client sticks to a specific server region until one of the following occurs:

- ▶ The client session is ended (that is, the cookie is deleted).
- ▶ The server instance is no longer available (that is, the CloneID is invalid).
- ▶ The Server Region is gone (that is, the WlmservletID is invalid).

4.3.2 HTTP session failover tests

We performed three HTTP session failover tests:

- ▶ Memory-to-memory replication with a single replica
- ▶ Memory-to-memory with the entire domain as replica
- ▶ DB2 persistence of the HTTP session

Memory-to-Memory replication with a single replica

In this test, the workload is driven to all three servers. We set `HttpSessionCloneId` for server A, B, and C to `h6sr01aSC49`, `h6sr01bSC50`, and `h6sr01cSC54`, respectively. We took down server A, which had `HttpSessionCloneId h6sr01aSC49`.

In the WebSphere Workload Simulator log shown in Example 4-5, notice that the failed sessions on server A were restarted on server C, which was Server A's single replica. Also notice that the `JSESSIONID` of the failover session had server C's `HttpSessionCloneId h6sr01cSC54` appended at the end.

When the request was sent over from the client again, the WebSphere plug-in HTTP Server parsed the last (rightmost) `CloneId` and used it to locate the server that held the HTTP session.

Example 4-5 WebSphere Workload Simulator log for HTTP session memory-to-memory single replica replication

```
....
06/13/05 17:08:00 - 0001: WAS Server H6SR01A is running session
JHvfgaUkd1AqULF19PkQ5Ym:h6sr01aSC49
06/13/05 17:08:03 - 0003: WAS Server H6SR01A is running session
IqhJbiEwiRdC_pNXU7BG16H:h6sr01aSC49
06/13/05 17:08:03 - 0002: WAS Server H6SR01A is running session
2u4D5PnIr3vrH3Cd2U0kr6_:h6sr01aSC49
06/13/05 17:08:03 - 0006: WAS Server H6SR01C is running session
0001NT0pjodewDHggUnQsJ2BIOH:h6sr01cSC54
06/13/05 17:08:04 - 0005: WAS Server H6SR01B is running session
0001JaLR7_dNjK0pkxTIbFLjqpj:h6sr01bSC50
06/13/05 17:08:04 - 0004: WAS Server H6SR01A is running session
jkSLmYIxvwpZQA1u-Ext0m:h6sr01aSC49
06/13/05 17:08:11 - 0007: WAS Server H6SR01C is running session
0001hI4Mge4vCr5k053p6kd17L2:h6sr01cSC54

06/13/05 17:08:37 - 0001: >>> Session
JHvfgaUkd1AqULF19PkQ5Ym:h6sr01aSC49:was interrupted on Server H6SR01A,
and restarted on server H6SR01C
```

06/13/05 17:08:37 - 0001: **WAS Server H6SR01C is running session**
JHvfgaUkD1AquLF19PkQ5Ym:h6sr01aSC49:h6sr01cSC54

06/13/05 17:08:38 - 0003: >>> Session
IqhJbiEwiRdC_pNXU7BG16H:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C

06/13/05 17:08:38 - 0003: **WAS Server H6SR01C is running session**
IqhJbiEwiRdC_pNXU7BG16H:h6sr01aSC49:h6sr01cSC54

06/13/05 17:08:38 - 0002: >>> Session
2u4D5PnIr3vrH3Cd2U0kr6_:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C

06/13/05 17:08:38 - 0002: **WAS Server H6SR01C is running session**
2u4D5PnIr3vrH3Cd2U0kr6_:h6sr01aSC49:h6sr01cSC54

06/13/05 17:08:38 - 0004: >>> Session
jkSLmYIxvwpZQA1u-Ext0m:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C

06/13/05 17:08:38 - 0004: **WAS Server H6SR01C is running session**
jkSLmYIxvwpZQA1u-Ext0m:h6sr01aSC49:h6sr01cSC54

...

Memory-to-memory replication with entire domain as replicas

In this test (see Example 4-6), we also took down server A. Notice that some sessions were failed over to server B, while other sessions failed over to server C.

Example 4-6 WebSphere Workload Simulator log for HTTP session memory-to-memory entire domain replication

...

06/14/05 11:20:32 - 0004: **WAS Server H6SR01C is running session**
0001pkr-70YbpPpBi5CnrHkOf1p:h6sr01cSC54

06/14/05 11:20:33 - 0007: **WAS Server H6SR01B is running session**
0001qMM-SItv2M88dGud47mh8kV:h6sr01bSC50

06/14/05 11:20:35 - 0002: **WAS Server H6SR01B is running session**
0001C_Y_p3CSXu97oi2yLQnEKxC:h6sr01bSC50

06/14/05 11:20:35 - 0003: **WAS Server H6SR01A is running session**
0001hBZhs2rJ3MkHXDRpmaiuvZP:h6sr01cSC49

06/14/05 11:20:37 - 0001: **WAS Server H6SR01A is running session**
0001Y-wcRs-6jDabMTOMEBgLCjL:h6sr01cSC49

06/14/05 11:21:14 - 0006: **WAS Server H6SR01A is running session**
PaFn65vBjIrS0d7aVSQS_lb:h6sr01aSC49

06/14/05 11:21:15 - 0004: WAS Server H6SR01A is running session
g2k8MZ7033ok0JE12XVQ89d:h6sr01aSC49
06/14/05 11:21:19 - 0001: WAS Server H6SR01A is running session
RRYD1x-12gti7FQ3VkJT0fdX:h6sr01aSC49
06/14/05 11:21:20 - 0005: WAS Server H6SR01A is running session
h2g01KP1JeTvKx-oSbW0oNy:h6sr01aSC49

06/14/05 11:21:27 - 0006: >>> Session
PaFn65vBjIrS0d7aVSQS_lb:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01B
06/14/05 11:21:27 - 0006: **WAS Server H6SR01B is running session**
PaFn65vBjIrS0d7aVSQS_lb:h6sr01aSC49:h6sr01bSC50

06/14/05 11:21:28 - 0004: >>> Session
g2k8MZ7033ok0JE12XVQ89d:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C
06/14/05 11:21:28 - 0004: **WAS Server H6SR01C is running session**
g2k8MZ7033ok0JE12XVQ89d:h6sr01aSC49:h6sr01cSC54

06/14/05 11:21:32 - 0001: >>> Session
RRYD1x-12gti7FQ3VkJT0fdX:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C
06/14/05 11:21:32 - 0001: **WAS Server H6SR01C is running session**
RRYD1x-12gti7FQ3VkJT0fdX:h6sr01aSC49:h6sr01cSC54

06/14/05 11:21:33 - 0005: >>> Session
h2g01KP1JeTvKx-oSbW0oNy:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C
06/14/05 11:21:33 - 0005: **WAS Server H6SR01C is running session**
h2g01KP1JeTvKx-oSbW0oNy:h6sr01aSC49:h6sr01cSC54

...

Persist HTTP session to DB2 table

See 8.3.2, “Setup for session persistence with DB2” on page 225 for how to set up HTTP session persistence with DB2. Again, to test session failover, we took down server A (see Example 4-7) and the sessions failed over to server B and C.

Example 4-7 HTTP session replication by DB2 persistence

...

06/14/05 14:00:50 - 0002: WAS Server H6SR01B is running session
-Exd2wnPxvFjDnjfjDT1s1h:h6sr01bSC50

06/14/05 14:00:53 - 0004: WAS Server H6SR01B is running session
jkNI dx2oIQtzAYv0eBd1mBR:h6sr01bSC50
06/14/05 14:00:55 - 0005: WAS Server H6SR01A is running session
EmUitUE1rKyS2Sycv4I_T_G:h6sr01aSC49
06/14/05 14:00:56 - 0006: WAS Server H6SR01C is running session
p216ShcH437AIGjN8odLm-k:h6sr01cSC54
06/14/05 14:01:34 - 0002: WAS Server H6SR01A is running session
7aTMnRXG0RFBCpyxYq0G7w4:h6sr01aSC49
06/14/05 14:01:36 - 0004: WAS Server H6SR01A is running session
ULZP5c3UE7n8-euuStAoMK9:h6sr01aSC49
06/14/05 14:01:38 - 0006: WAS Server H6SR01A is running session
Rrf1SWtD800B4Hv6l-vB0Ar:h6sr01aSC49
06/14/05 14:01:38 - 0005: WAS Server H6SR01C is running session
v2K-61f_kQF_Opj05RDfMtr:h6sr01cSC54
06/14/05 14:01:41 - 0001: WAS Server H6SR01A is running session
CBicAt8Eo3hIQsgiTXMc9H9:h6sr01aSC49
06/14/05 14:01:42 - 0003: WAS Server H6SR01C is running session
8jvyOLgehN53ZYgqHMvH1Ro:h6sr01cSC54
06/14/05 14:01:47 - 0007: WAS Server H6SR01A is running session
Dmzlr_f7rdQNAZrk0GpEOcN:h6sr01aSC49
...
06/14/05 14:02:02 - 0002: >>> Session
7aTMnRXG0RFBCpyxYq0G7w4:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C
06/14/05 14:02:02 - 0002: **WAS Server H6SR01C is running session**
7aTMnRXG0RFBCpyxYq0G7w4:h6sr01aSC49:h6sr01cSC54

06/14/05 14:02:02 - 0006: >>> Session
Rrf1SWtD800B4Hv6l-vB0Ar:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01C
06/14/05 14:02:02 - 0006: **WAS Server H6SR01C is running session**
Rrf1SWtD800B4Hv6l-vB0Ar:h6sr01aSC49:h6sr01cSC54

06/14/05 14:02:03 - 0004: >>> Session
ULZP5c3UE7n8-euuStAoMK9:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01B
06/14/05 14:02:03 - 0004: **WAS Server H6SR01B is running session**
ULZP5c3UE7n8-euuStAoMK9:h6sr01aSC49:h6sr01bSC50
...

4.4 HTTP sessions versus stateful session beans

There are multiple ways to maintain a user session. But people are often confused by the difference between HTTP sessions (how servlets maintain state) and stateful session beans. To help clarify this issue, we compare both and discuss how they can work together.

Servlets

- ▶ They are called using the HTTP protocol.
- ▶ They can be an EJB client.
- ▶ They implement mostly input processing and presentation functionality.
- ▶ They implement only limited business logic and simple data access logic.

Session beans

- ▶ They represent a single client inside the J2EE server.
- ▶ They are called using RMI over the IIOP protocol.
- ▶ They do not implement presentation logic.
- ▶ They implement or wrap business logic.
- ▶ They implement simple access to data.

Managing client state in a servlet

A servlet client is always an HTTP client, and HTTP is a stateless protocol. You can use different approaches to get information about the client and maintain that information about the session. These pieces of information are generally related to customer input (or to any other client-related presentation data). Common methods of getting this information are by the use of cookies, or via URL rewriting.

If cookies are enabled, the Session Tracker will use a unique Session ID to match user requests with their HttpSession objects on the server. If you are using cookies, the browser must support cookies and must also allow cookies. Session cookies are not stored in a file; they simply live in the memory of the browser. In some browsers, you can explicitly allow session cookies and not allow the use of other cookies.

With URL rewriting, all links that you return to the browser or redirect have the session ID appended to them.

The servlet specification defines an API that provides HTTP session state semantics. Using this API creates a session object in which the HTTP session data is stored. WebSphere Application Server for z/OS and OS/390 supports maintaining this session object in a shared DB2 database, as well as in memory. For a discussion about the performance issues of both options, see 4.2.3, “Session affinity” on page 74.

Managing client state in a stateful session bean

The client of a session bean is always a Java program talking RMI/IIOP with the session bean; it can be an applet, a servlet, another EJB, or any other Java program.

There are two types of session beans, stateless session beans and stateful session beans, which are described as follows:

- ▶ A *stateless* session bean does not maintain a conversational state for a particular client.

When a client invokes the method of a stateless bean, the bean's instance variable may contain a state, but only for the duration of the invocation. When the method invocation is finished, the state is no longer retained. Stateful sessions can support multiple clients.

- ▶ A *stateful* session bean maintains a conversational state with the client.

On the EJB tier managing state is done through the use of stateful session beans. These beans have state or instance fields that can be initialized and changed by the client at each method invocation. This state can be used to process business methods invoked by the client.

The state is retained for the duration of the client bean session. If the client terminates, the session ends and the state disappears. Stateful sessions support just one client. A stateful session bean may hold such information as the shopping cart of a customer in an e-commerce application.

By using stateful session beans you get all the benefits of an EJB component, including the following:

- Lifecycle management
- Concurrency control
- Synchronization
- Declarative Transactions
- Security Management

These advantages are not so pronounced under WebSphere because HTTP session management also includes concurrency control, synchronization options, and security. Therefore, the choices whether to use stateful session beans revolve more around the overall architecture that you employ. Stateful session beans are available to Web clients and non-Web clients. If you do not have a Web tier or if you want both Web and non-Web clients to reuse stateful business logic, then stateful session beans should be considered.

How a client finds its stateful session bean

In the following simple example, we explain how a servlet, holding an HTTP session, and a stateful session bean work together if a client is doing a transaction on a J2EE application.

A typical J2EE application consists of multiple servlets, JSPs, and EJBs (session beans and entity beans). The client, if it is an HTTP client, will connect to the servlet, and the servlet will create an HTTP session. The session data is stored in a session object. A session ID is associated with this session object. The servlet then interacts with a session bean to run the business logic.

In order to interact with the EJB, the servlet does a lookup to the home interface of that EJB. After getting the home interface, the servlet issues an `ejbCreate` in order to instantiate the EJB object. As the result of the `ejbCreate`, the container sends an EJB handle to the client.

An *EJB handle* is a reference to that EJB. The handle contains information about in which application server instance, and in which servant of that instance, this EJB lives. EJB local reference concept was introduced in EJB specification 2.0. Local reference refers intra-JVM lookup of EJB. It allows more efficient EJB lookup and method invocation.

Using this handle, the servlet will always find the stateful session bean. The same is true for other EJB clients, but in this case we want to show the interaction between servlets and stateful session beans in an HTTP session.

The most logical place to store this EJB handle is in the `HttpSession` object, because this results in an easy correlation between `HttpSession` and the stateful session bean; see Figure 4-8.

Now you have to make sure that there is a correlation between your client and the `HttpSession`. This is done using session affinity in your infrastructure.

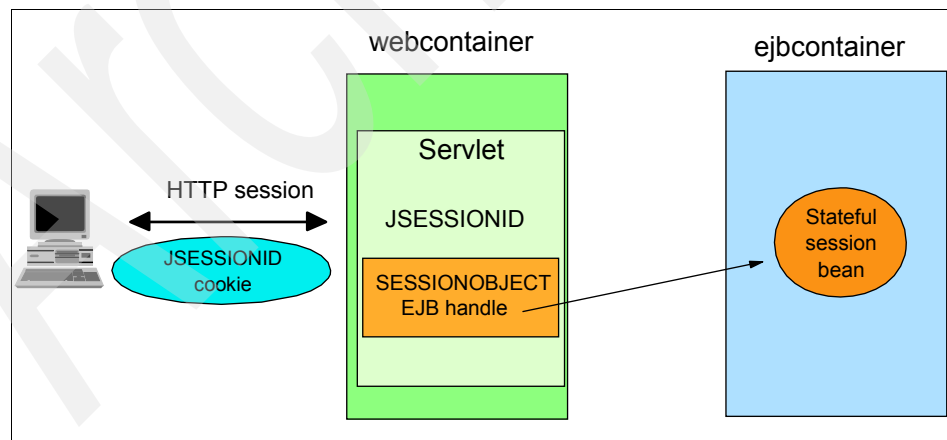


Figure 4-8 Correlating the `HttpSession` with stateful session beans

The failover of a stateful session bean is similar to HTTP session failover. The J2EE 1.4 specification requires the HTTP session state object to be able to contain local references to EJBs. WebSphere Application Server V6 can collate stateful session bean replica and HTTP session replica with hot failover.

Archived

Archived

Architectures for availability

Have you ever gone to a Web site and had the browser time-out? Perhaps you filled in a form, and then didn't know if the transaction actually worked or not—and if you did a refresh, you may have wondered: was the transaction executing for the first time, or for the *second* time?

Such scenarios result from poor availability or poor scalability. In designing an e-business infrastructure, there's a fine balance to be achieved between availability, scalability, security, and cost. There are several options available to help you design the best infrastructure for your needs.

In this chapter we describe the options for creating highly available system architectures (or infrastructures). Obviously there are many ways to architect a system to achieve high availability, and we do not document them all in this book. However, we do present architectures that we feel represent the most common ways to achieve high availability.

Important: When you design a system, you may want to consult the Patterns for e-business Series Redbooks. They describe a systematic approach from the high-level Business Patterns (a given business interaction between users, businesses, and data) down to Runtime Patterns and Product Mappings (what the major infrastructure components are, how they interact, and what products you can use to build the systems).

5.1 Things to consider

Availability

The most common way to achieve availability is by providing redundant components. This availability requirement includes networks and network appliances (firewalls, routers, and switches), and redundant Internet Service Providers. The objective is to avoid single points of failure and to provide sufficient bandwidth to handle the maximum projected workload.

The requirement for redundancy is also based on the reliability of the hardware platform chosen, in conjunction with the base operating system chosen. In many cases you need more than two servers to handle the workload, as well as to solve the availability issue.

Scalability

The scalability of an infrastructure and application server environment can be accomplished either horizontally or vertically. You can have many small machines, or a few large machines. However, once you move beyond having one server, a secondary issue must also be addressed: coordination of work and data across multiple servers.

The zSeries hardware platform is unique in that it allows you to accomplish both horizontal and vertical scaling on the hardware. This can be accomplished by adding engines, by using the LPAR capabilities, VM Linux guests, and also by using the Parallel Sysplex technology of zSeries.

Efficiency

In order to minimize system failure of redundant components, you should also consider how to make each component of the architecture as efficient as possible.

For instance, a back-end application server may failover because the CPU of the system is unable to keep up with a large spike in customer demand. Some of the content that the customers are going after may be static, and by utilizing proxy servers, you could cache the static content on the front end.

This will help to keep “unnecessary” traffic out of the application server region, thereby making the system perform more efficiently and less likely to suffer bottlenecks leading to failures.

Security

The most secure system is a closed system. As you add more layers to an application, and add redundancy and failover in the infrastructure and network, building a secure end-to-end system becomes a more complicated task.

Cost

The more infrastructure you have, the higher the cost. The question then becomes, is the increase in cost justified by the increase in availability and scalability. These functions of a system design not only provide failover, but also improve customer satisfaction by reducing or eliminating failures. This can reduce customer support problems and may actually save money.

The following architectures focus on using the zSeries hardware running z/OS in order to provide the highest level of availability in a WebSphere Application Server production environment. The zSeries hardware platform and the z/OS operating system, functioning in a Parallel Sysplex configuration, provide the highest level of failover and availability at this time.

The two basic types of architectures are two-tier and three-tier. It is important to define some of the terminology before reviewing the architectures. The easiest way to do this is to use the basic application model as a reference point. You can find this application model in 1.3, “J2EE application characteristics” on page 10.

5.2 Logical tiers

For the purpose of these architectures, a tier is any system that participates in the actual application transaction. A firewall, switch, router, sprayer, or simple HTTP server acting as a reverse proxy is *not* considered one of the tiers. It can be viewed as part of the networking infrastructure.

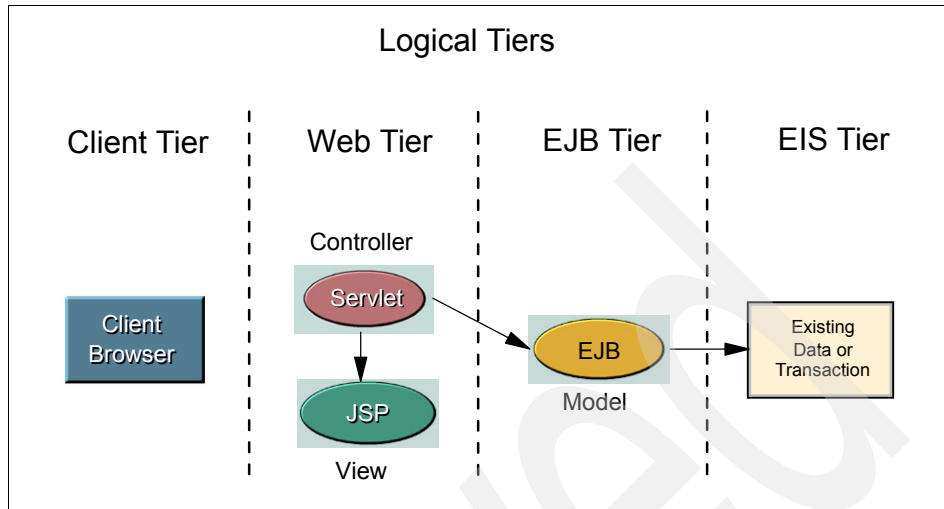


Figure 5-1 Tiers in the J2EE application model

The J2EE architecture, as shown in Figure 5-1, describes several logical tiers. These relate to the function provided by those tiers:

- ▶ Client tier (Web browser or fat client)
- ▶ Web tier (HTTP server, presentation layer, and controller function)
- ▶ EJB or Enterprise Java Bean tier (business logic and some data access)
- ▶ EIS or Enterprise Information System tier (back-end existing data and transactions)

5.3 Logical architectures

In this section we review how the multitier J2EE architecture can be translated to two-tier and three-tier logical environments.

5.3.1 Two-tier logical architecture

The two-tier architecture is the simplest. When dealing with static content, pages, and pictures, it is the easiest to implement and support. In a two-tier architecture for dynamic e-business applications, the Web, EJB, and EIS logical tiers reside on the same server; see Figure 5-2.

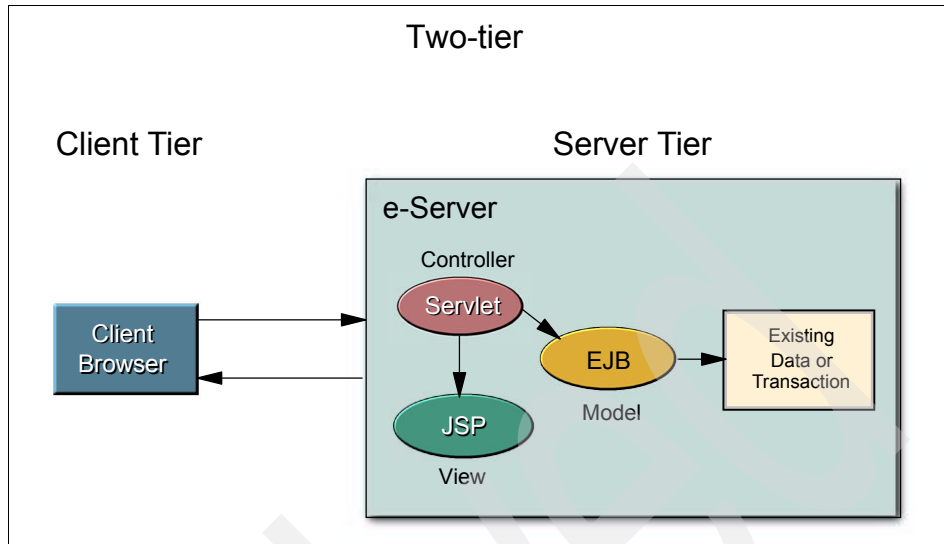


Figure 5-2 Two-tier logical architecture

5.3.2 Three-tier logical architecture

In a three-tier architecture, there are at least two options. The first is to place the Web and EJB components on the same server, as a middle tier. The EIS component would then reside on a third or backend tier or server.

The second option is to place the Web tier (servlets and JSPs) on the middle tier and the EJB and EIS tier on the backend server. This option isolates the presentation layer from the business logic and data access. This solution is sometimes referred to as a distributed application model.

In the current J2EE application server environment, this can be done using Web Services or Message Driven Beans as the connection between the middle and back-end servers.

In the case of a zSeries solution, the middle tier and back-end tier could both reside on the same physical hardware, but still be physically isolated as logical partitions or LPARs, completely separated from each other. Figure 5-3 is an illustration of a three-tier architecture.

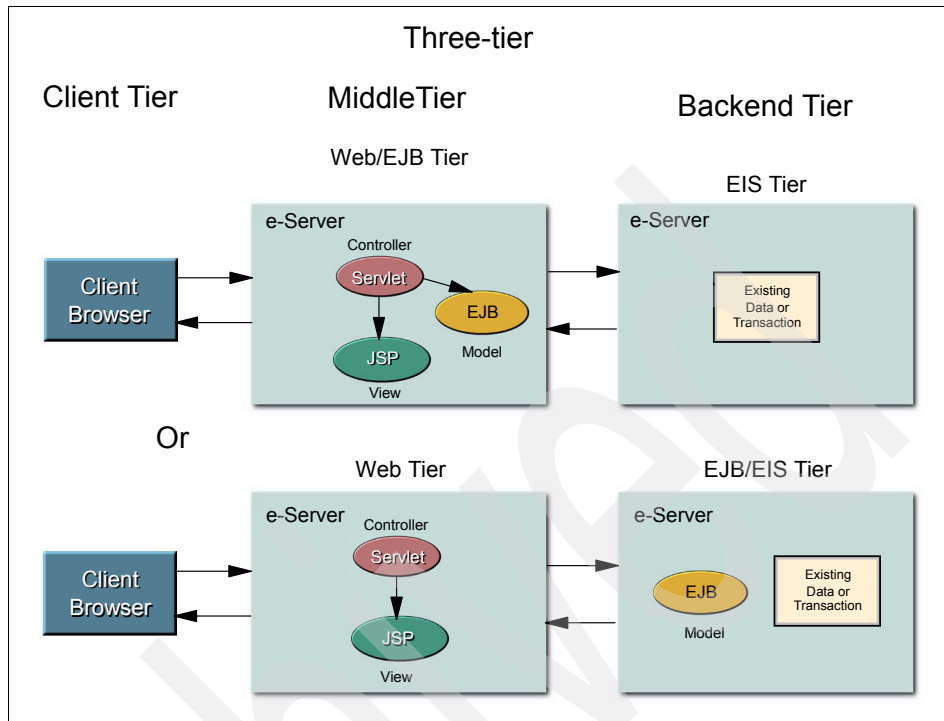


Figure 5-3 Three-tier logical architecture

5.4 Physical architectures

In this section we provide examples of physical architectures that provide the infrastructure within which the logical layers can be mapped. How you structure your tiers has a major impact on scalability, availability, and performance.

The following guide may help with deciding the infrastructure to use:

- ▶ Analyze and clearly define your requirements for the infrastructure, such as availability level, acceptable recovery process, performance goals, scalability, and capacity plans.
- ▶ Understand all the concepts that need to be considered for each of these requirements.
- ▶ Plan the infrastructure in a diagram. Initially show the proposed infrastructure in a basic form and then scale it up to make sure that it scales well. Such a diagram is useful for identifying single points of failure, scalability problems, and possible bottlenecks in the infrastructure.

- ▶ Make sure you have well-defined and scalable naming standards for all aspects of the infrastructure.
- ▶ Create a prototype infrastructure and test all the requirements outlined during analysis.

In general, the more physical tiers that you have, the more flexible the architecture will be. Decoupling components on different tiers can allow components to be changed or replaced while minimizing the impact on the other tiers.

Workload balancing can also be enhanced in multi-tier environments with more points for workload balancing decisions to take place. However, by increasing the number of physical tiers you are including more remote calls into the infrastructure. This adds more network activity as well as the CPU overhead related to marshalling or serializing requests to and from Java Objects. More tiers also raise more security concerns since the request flows between different environments and security managers.

How many physical tiers to use and what layers should be mapped to them can be a difficult decision. The following sections describe some common scenarios in detail to help show the advantages and disadvantages with using different infrastructure models.

5.5 z/OS base infrastructure

If we are to build a WebSphere Application Server infrastructure that is scalable and highly available, then the underlying z/OS infrastructure and Enterprise Integration Subsystems (EIS) also need to be scalable and highly available. On z/OS this means making full and redundant use of Parallel Sysplex technology.

The Parallel Sysplex should be set up with at least two LPARs to provide redundancy for failover situations. Each aspect of the Parallel Sysplex architecture should contain redundancy. This includes the Couple DataSets, Coupling Facilities, Coupling Facility structures and Coupling Facility Links. Refer to *z/OS V1R4.0 MVS Setting Up a Sysplex*, SA22-7625 for a detailed explanation of what is involved with configuring an available base infrastructure.

All subsystems connected to the WebSphere Application Server infrastructure should also be configured with redundancy. This can be achieved by creating redundant subsystem servers on different LPARs or by configuring the subsystems as sysplex-aware.

Sysplex-aware subsystems, such as DB2 in Datasharing Mode, CICS in a CICSplex configuration, and WebSphere MQ using Shared Queues, can take

advantage of z/OS advanced workload management concepts, as well as providing availability during failover situations.

WebSphere Application Server, when configured in Network Deployment mode, can take full advantage of the underlying z/OS Parallel Sysplex and Workload Manager capabilities to provide for fully available and scalable infrastructures.

Figure 5-4 shows a simple application server cluster configured over two LPARs in a sysplex. If one of these LPARs were to go down, the cluster could still process requests through the remaining server clone.

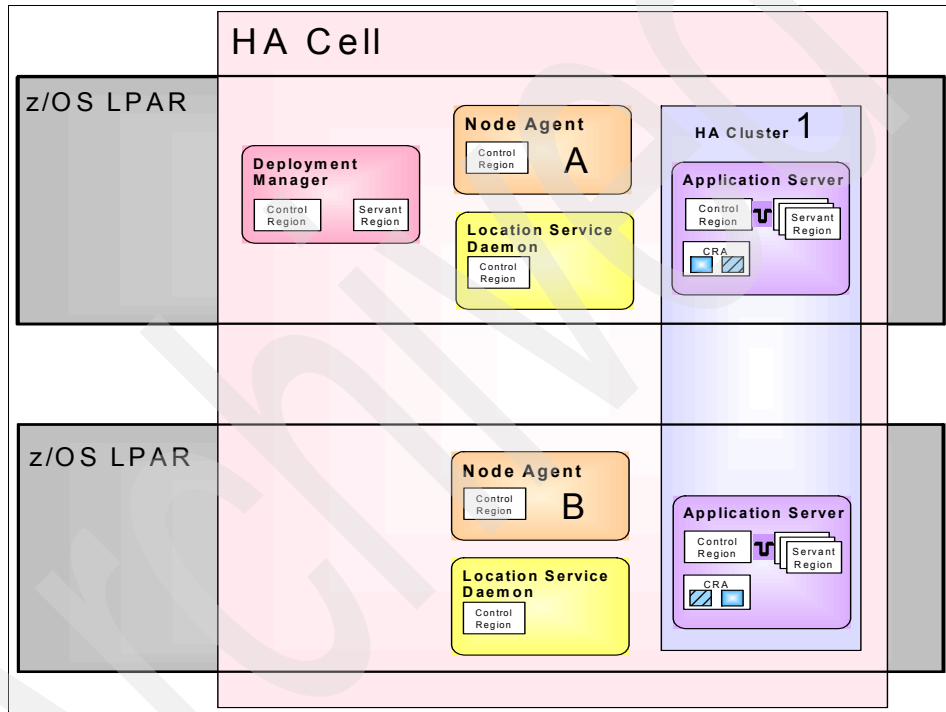


Figure 5-4 WebSphere Application Server cluster utilizing the zSeries infrastructure

5.6 Two-tier physical architecture

The two-tier architecture is not new to z/OS. The traditional connection to a z/OS system (OS/390 and MVS) has been a 3270 terminal connected through a control unit to a z/OS system running an IMS or CICS application. The e-business model changes the presentation interface, but much of the business logic and data access still remain in CICS or IMS.

The two-tier infrastructure is the simplest to set up and support.

In this architecture there is usually an HTTP server in front of the application server. This is true for any eServer™ solution. WebSphere Application Server for z/OS can be frontended by the IBM HTTP Web server, either running on the z/OS platform or on another platform such as Linux.

The HTTP Web server uses a plug-in, which acts as an agent, redirecting requests to the HTTP Transport components of the application servers.

Reasons for using a separate HTTP Web server:

- ▶ You can separate out the HTTP server (Web server) and locate it within an untrusted network. You can then place the actual application server behind a second firewall. This allows you to secure the business logic and data.
- ▶ Using the HTTP Web server plug-in (rather than allowing direct HTTP access to the HTTP Transport) provides workload management and failover capabilities by distributing requests evenly to multiple application servers and routing requests away from failed application servers.
- ▶ By allowing static content to be served and cached on the HTTP Web server reduces the requests sent to the application servers for processing.

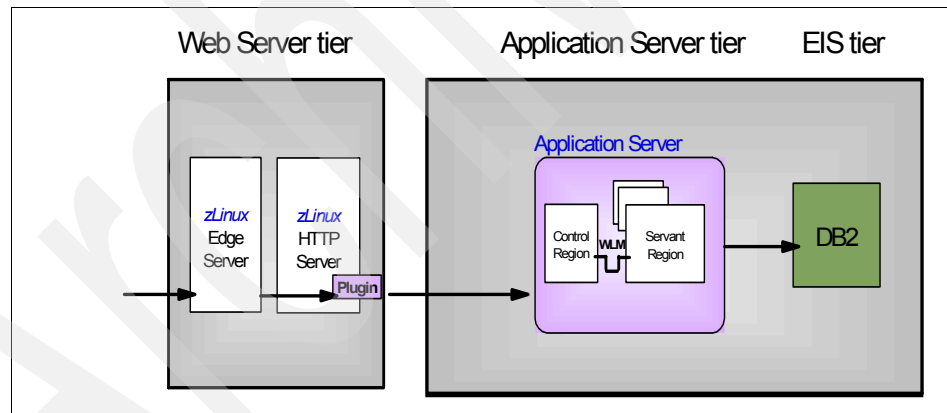


Figure 5-5 Two-tier physical infrastructure

The basic two-tier infrastructure shown in Figure 5-5 shows the HTTP Web server connected directly to the HTTP Transport listener of the application server control region (CR). The control region routes the request to the appropriate servant region (SR) based on the Workload Manager policy.

This configuration could be a simple base application server or one application server in a Network Deployment cluster. Servant regions are started as required

by the Workload Manager subsystem, thus providing vertical scaling behind the control region on the LPAR.

This setup is limited in its failover and recovery since there is no alternative application server that can process workload in the event of failure. There are various single points of failure in the diagram: the HTTP server, the z/OS system, the application server, and DB2. Any failure in these systems will cause a complete outage of the application.

Two-tier with high availability

For a highly available solution we need to identify and remove all single points of failure from the infrastructure. By building redundancy into the configuration we are providing alternative sets of resources to process requests if a part of the infrastructure goes down. As well as availability, this scaling of the environment provides for workload balancing opportunities at various stages.

In Figure 5-6 on page 109 the use of an Edge Server in front of the HTTP server acts as a sprayer or workload balancer to the HTTP servers, distributing the HTTP requests among them. In the case of an HTTP server failure, all the requests would be funneled through the remaining HTTP server.

You will also notice the addition of a Dynamic Virtual IP Address (DVIPA) for the HTTP Transport listeners on the control regions. This provides additional workload management for all non-affinity requests from the HTTP plug-in. Affinity base requests are managed by the plug-in on the HTTP server.

In order to make the two-tier architecture highly available, the WebSphere Application Server infrastructure needs to be in Network Deployment mode. Network Deployment brings in the concept of cells, which are administration domains that control the configuration and coordination of resources.

Network Deployment allows application servers to work together in logical application server clusters. A cluster can comprise application servers on multiple nodes on different LPARs. The application server instances are called clones. Node Agents run on each node and provide the synchronization required to maintain the application servers' configurations.

It is recommended that you have at least two application clones spread over two nodes on different LPARs. It is possible to have two nodes on the same LPAR, but this would provide only limited failover.

Spreading the cluster over different LPARs will provide enough redundancy to carry on processing workloads if one node or LPAR fails. Furthermore, it would allow you to recycle a server clone after configuration changes without taking the whole cluster down.

A fully redundant two-tier architecture is depicted in Figure 5-6. In this case, each service or system is redundant to support almost any failover condition. This two-tier architecture is commonly seen on the z/OS platform.

The application server tier resides on two nodes spread over two different LPARs in a Parallel Sysplex. A failure of one of these nodes will leave the servers on the other nodes able to continue processing the workload and also provide a system from which to restart the failing application server in *Peer Restart Recovery* mode if required.

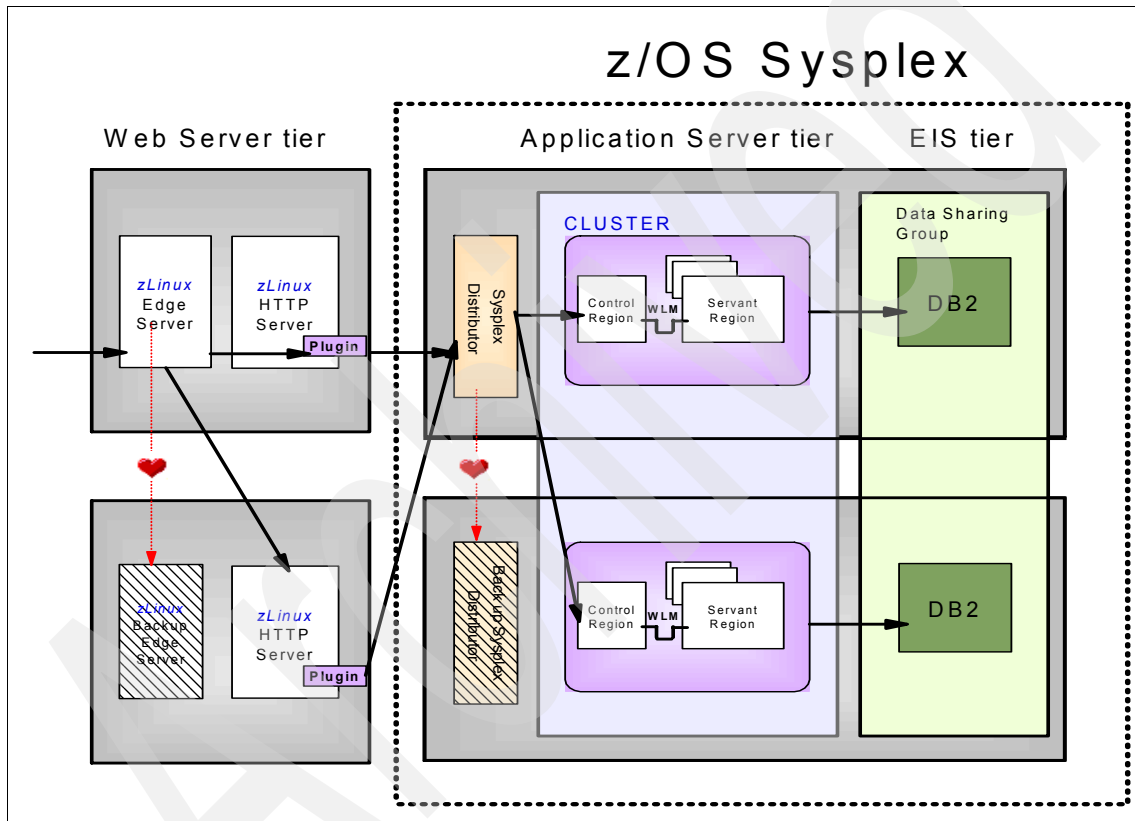


Figure 5-6 Two-tier infrastructure with availability

Resources such as DB2, CICS, and IMS are configured across the sysplex, for example, DB2 in DataSharing mode and CICS set up in a CICSplex® SM configuration. If one of the LPARs goes down, then these resources have subsystems on other LPARs that can continue to process the workload.

Important: This level of sharing is difficult to achieve on any other platform.

In most non-zSeries architectures, a three-tier solution is used, with the database or transaction server on a third or back-end tier.

J2CA resources

J2EE Connector Architecture, also known as J2CA, defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EIS), mainframe transaction processing applications, database systems, and legacy applications not written in the Java programming language.

By defining a set of scalable, secure, and transactional mechanisms, J2CA enables the integration of EISs with application servers and enterprise applications. J2CA enables an EIS vendor to provide a standard resource adapter for its particular system. The resource adapter plugs into an application server, providing connectivity between the EIS, the application server, and the enterprise application.

By accessing EIS subsystems on the same LPAR as the WebSphere Application Server you get all the benefits of “local” connections through the J2CA Adapter. In this configuration you best exploit the advantages of the z/OS platform.

▶ DB2

A DB2 DataSharing group comprises DB2 subsystems spread over multiple LPARs in a Parallel Sysplex. WebSphere Application Servers can use the DB2 RRS JDBC™ Provider to access the DB2 subsystem. DB2 comes with a type 2 JDBC/SQLJ driver, which provides full two-phase commit transaction support based on Resource Recovery Services (RRS). Access to DB2 is usually through a J2EE Data Source but is included here for completeness.

▶ CICS

The CICS Transaction Gateway (CTG) provides the ECI Resource Adapter that is a J2C-compliant connector to CICS. When the WebSphere Application Server is on the same LPAR and the ECI Adapter is running in local mode, then performance is much better because cross-memory communication is used rather than TCP/IP. The ECI Resource Adapter is a one-phase capable resource adapter. However, when used through WebSphere Application Server in local mode, two-phase capability is available through RRS.

Note: In WebSphere Application Server V6 there is a new JMS provider called the *default messaging provider*. This JMS provider is a fully J2CA 1.5-compliant resource adapter, and it is fully integrated into WebSphere Application Server. For further details, see Chapter 6, “WebSphere platform messaging” on page 123.

Security

Security needs to be included at every step in defining an infrastructure. The three main objectives of a security model are authentication, access control, and auditing. All three of these objectives should be accomplished by the infrastructure, if possible.

The application should be the last choice for accomplishing any of these. The reason for this is to ensure that any change in the users or use of the application will result in external modifications, and no need to change the application itself.

Authentication

In the two-tier architecture shown in Figure 5-6 on page 109, authentication can be done in two ways:

- ▶ The outbound HTTP server and the caller's identity can be attached to the request as part of the HTTP header. The WebSphere Application Server could then use a Trust Association Interceptor to map the passed identity to a z/OS System Authorization Facility (SAF) identity, and then validate access to the application and run the request under that identity.
- ▶ The second approach is to set up and configure the WebSphere Application Server to provide the authentication and access control. This can be done using forms-based authentication or basic authentication. Once a user is authenticated, WebSphere Application Server will determine if the user has the authority to access the resource.

Authorization

The current J2EE specification provides for role-based access control on the classes and methods within an application. This means that once the identity of the user has been established, as described above, this identity is mapped to a role and access to each component is checked to ensure that the role the user is operating in has the authority to access the class or method.

These roles are defined in the deployment descriptor.

The J2EE specification does not provide for instance-based access control, where authorization is checked against the particular instance of a class. This should currently be done using either a rule engine or programmatic control.

WebSphere Application Server security is integrated into z/OS through the use of the SAF interface, backed by a security subsystem such as RACF®. Authorization and identity management should be carefully reviewed and configured to provide the required level of control.

Audit

One advantage to using WebSphere Application Server on z/OS is the audit capability: the z/OS system can be configured to generate SMF records. This will give you track information about requests that are being processed, including the userid that the request is being executed under.

J2CA resources security

The standard methods of providing credentials to EIS subsystems in WebSphere Application Server V6 is through the use of predefined J2CA Authentication Aliases in the container (`res-auth=container`), or by passing the userid and password programmatically by the application (`res-auth=application`). Both of these options require security information to be kept outside of RACF.

For locally connected resources such as the DB2 RRS JDBC Provider and the CICS ECI Resource Adapter in local mode you can use Thread Security (also known as `SynchToOSThread` under WebSphere Application Server V4). When `SynchToOSThread` is set on in the Global Security options, then (unless overridden with an authentication alias) WebSphere Application Server flows the currently authenticated principal userid to the EIS subsystem, which will then be authorized against RACF. This provides a much stronger method for passing credentials to the EIS subsystems for RACF checking than the standard J2CA Authentication Aliases.

Local performance vs. workload balancing

Although local connections to EIS subsystems provide for better security and performance, there is a downside. When accessing an EIS subsystem locally you are not workload managing this connection. Therefore, if the local subsystem goes down, your connection is not going to be routed to another subsystem in the sysplex.

If you need workload balancing between WebSphere Application Server and the EIS subsystems, you may want to consider using a remote connection using Sysplex Distributor.

5.7 Three-tier physical architectures

Three-tier architectures provide a high level of decoupling. Which logical layers to decouple will depend on your requirements and current infrastructure. Two common three-tier architectures are to separate the Web/EJB tier from the EIS or to separate the Web from the EJB tiers. Each of these scenarios is discussed separately in the following sections.

5.7.1 Decouple Web/EJB and EIS layers

This scenario is common when the J2EE application is used to access existing data or transactions on a production z/OS system. In this configuration, the middle-tier server could be WebSphere Application Server on a non-z/OS or z/OS system.

If a z/OS system is used for both the middle and back-end tiers—and they are both in the same Parallel Sysplex—then there are several benefits, including better security and higher quality of service. However, it is not common to separate the middle tier from the EIS tier in a Parallel Sysplex unless you have specific architectural or operational reasons to do so.

Figure 5-7 shows an example of a three-tier architecture. You will notice that the main difference between the two-tier and the three-tier is that the new WebSphere Application Server workload has been placed on a separate server from the traditional legacy environment.

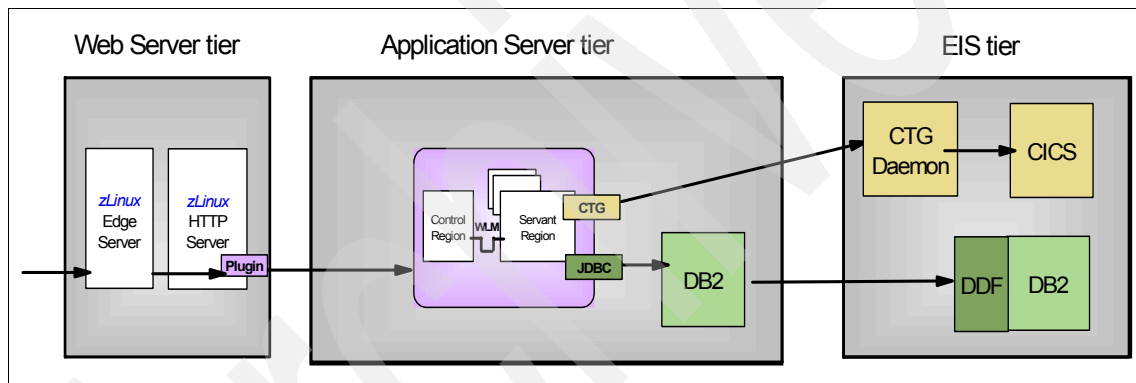


Figure 5-7 Three-tier infrastructure

Remote connection mechanisms

Each subsystem type has a set of mechanisms and protocols that can be used to connect to it remotely. Although it is beyond the scope of this book to go into detail on each of these, here is a short summary of what is available.

Remote access to DB2

- ▶ DB2 subsystem acting as an Application Requestor (AR)

The local z/OS DB2 subsystem will route requests to a remote z/OS DB2 subsystem on another LPAR using Distributed Relational Database Architecture™ (DRDA®) to a Distributed Data Facility (DDF) address space.

Although there is the overhead associated with using DRDA, this is the most highly available option when connecting remotely from a z/OS LPAR to DB2

on another z/OS LPAR since it can make full use of workload management controlled balancing of the target Data Sharing group, which can be accessed through a DVIPA and sysplex distributor.

This type of connectivity is also very secure since the local SAF can be used to authorize the access based on any of the locally available methods, such as J2CA credentials or Thread Security.

- ▶ **DB2 Connect™ EE**

Using a local DB2 type 2 JDBC driver to a DB2 Connect server (either local or remote), which will connect to the target DB2 subsystem. DB2 Connect uses the DRDA protocol to access the DB2 DDF address space. On the first connection a DRDA server list is requested, which provides an access list for all of the members of the DB2 data sharing group, sorted in Workload Manager preference. DB2 Connect will connect to the first member in the list. If this connection fails, then it will attempt to connect to the next and so on.

A user ID and password can be passed to the DB2 Connect server and will be flowed to the target DB2 subsystem to be authorized against z/OS SAF. Thread Security cannot be used with the DB2 Connect server.

- ▶ **Type 4 JDBC driver**

The new IBM DB2 Universal Driver for SQLJ and JDBC provides a type 2 and type 4 capable driver. The type 4 driver is a 100% pure Java connector that uses the DRDA protocol to access a remote DB2 from any Java-based system.

The type 4 driver is flexible, but it does not yet support the same level of sysplex awareness as the DB2 Connect server. The DB2 Universal Driver will be supplied with DB2 for z/OS V8. It can be obtained for DB2 V7 with the PTF for APAR PQ80841.

A user ID and password can be passed with the Datasource defined for the JDBC driver and will be flowed to the target DB2 subsystem to be authorized against z/OS SAF (refer to J2CA security earlier). Thread Security cannot be used with the type 4 driver.

For a more detailed look at DB2 connectivity, refer to *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435.

Remote access to CICS

- ▶ The CICS Transaction Gateway (CTG) is the strategic J2CA connector for accessing CICS from WebSphere Application Server. In remote mode a TCP/IP connection is established to a CTG daemon running on the same LPAR as the target CICS region. This daemon uses the External CICS Interface (EXCI) to pipe requests to the CICS regions. On z/OS the CTG daemon does not have any inherent workload management mechanisms.

- ▶ A common way to configure the CTG for CICS access is to go through a CICS Listening Region. WebSphere Application Server is configured to use the CTG in local mode to the CICS Listening Region on the same LPAR. This region listens for requests on the EXCI protocol from the CTG. These requests are passed to the CICS program designated in the CTG configuration, which can then forward the request to an Application Owning Region (AOR) for processing.

In a CICSplex SM configuration this request is balanced over all participating AORs using Distributed Routing. This makes for high availability since Workload Manager plays a part in making sure that the routing makes it to an available and suitable server to process the request.

The use of Thread Security is not supported through the CTG, so a userid needs to be passed to the ECI Resource Adapter because CICS does not carry out further authentication for requests received through the EXCI protocol. This may be a security concern for your infrastructure depending on the level of trust you have between your application servers and the CICS regions. CICS MRO Bind RACF profiles can be used to restrict which z/OS-based servers can access the CICS region.

- ▶ Since CICS TS 2.1, WebSphere Application Server can make use of the IIOP protocol to call EJBs running in the CICS EJB. Stateful or stateless sessions can be run within the CICS container to receive remote IIOP requests from WebSphere Application Server components. From the EJB components in CICS you can use the JCICS API to link to legacy CICS transactions and programs. Initially the EJB container in CICS had performance difficulties, but CICS TS 2.3 has made dramatic moves in this area, making the use of IIOP a feasible option for connecting to CICS from WebSphere Application Server.
- ▶ Since the communication for all of these options is at some point going across TCP/IP, you may also want to consider encrypting the transport using SSL. By using SSL V3 client authentication between WebSphere Application Server and the CICS region, you will be establishing the trusted relationship and secure the transport. Both the CTG and the CICS EJB Container support SSL encryption and mutual authentication using x509 certificates, providing full confidentiality on the connection.

For more information on the CTG, refer to the IBM Redbook *CICS Transaction Gateway V5 The WebSphere Connector*, SG24-6133. For information on the capabilities of the CICS EJB Container, refer to *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284.

Three-Tier - EIS decoupled with high availability

The three-tier architecture is no different from the two-tier architecture when trying to create a highly available solution. Redundancy and failover support are the keys to availability.

Just like the two-tier solution, the best way to handle a failure is to have redundancy and recoverability. In a zSeries and z/OS configuration, this can be accomplished using multiple LPARs in conjunction with multiple zSeries servers.

The following choices are possible.

- ▶ A single zSeries server partitioned into four LPARs, but a single Parallel Sysplex. This does not provide failover if the entire zSeries server fails.
- ▶ Two zSeries servers, each partitioned into two LPARs, but a single Parallel Sysplex. Each zSeries server contains an LPAR that contains a WebSphere Application Server in a Server Cluster, and a second LPAR that contains the traditional production CICS, DB2, or IMS server.

This is the best choice. In this configuration, if you lose an entire zSeries server, you still have an instance of every component to handle the failover.

- ▶ Two zSeries servers, each partitioned into two LPARS. One zSeries server has the middle tier, with two LPARs, each containing a WebSphere Application Server in a Server Cluster. The second zSeries server has two LPARs, each containing an instance of the existing production CICS, DB2, or IMS system.

The issue with this configuration is that if you lose a complete zSeries server, you lose both instances of WebSphere Application Server on the production server.

- ▶ Two or more zSeries LPARs in a sysplex, each containing WebSphere Application Server and the EIS subsystems in their relevant sysplex modes.

This is not, strictly speaking, a separation of the application server and EIS tiers, but it does provide for remote workload balanced connections between the application servers and the EIS subsystems, providing a highly available solution.

By decoupling the EIS tier, the J2CA connections to the EIS subsystems will become “remote”. In doing so you lose much of the benefits of local connections. Most notably, remote connections do not perform as well as the locally connected resources since a network connection and suitable resource protocol are required.

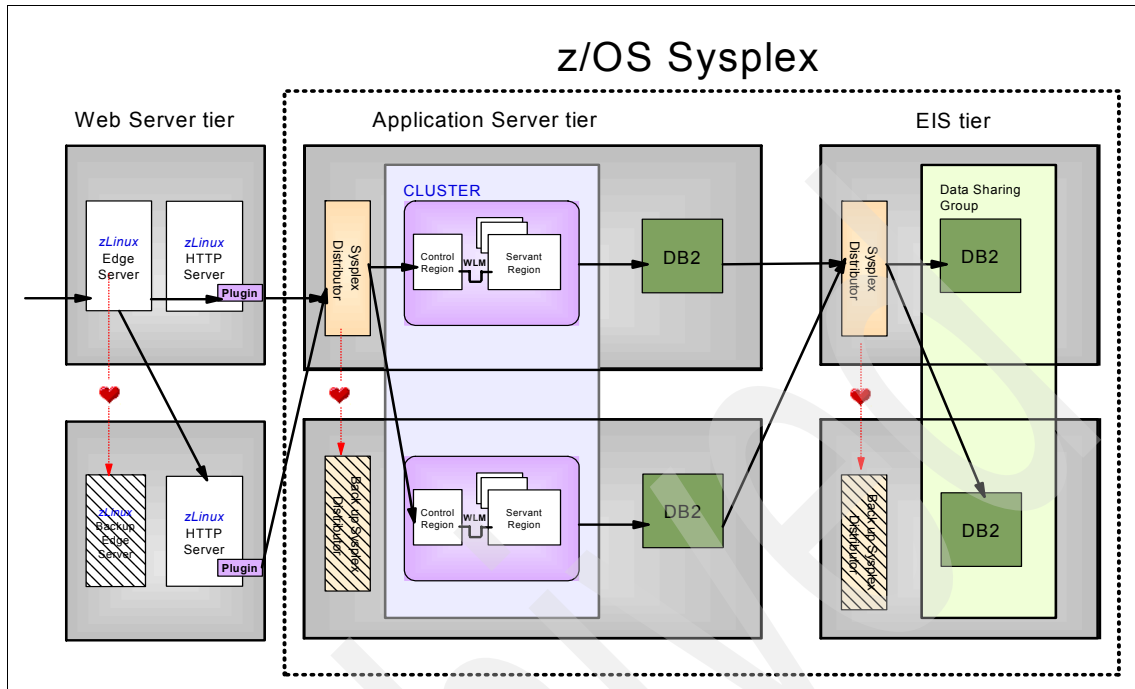


Figure 5-8 Three-tier architecture with failover support for DB2

Figure 5-8 shows a high-availability three-tier solution where the EIS tier is remote to the EJB tier.

Here the application servers are accessing DB2 through a local DB2 subsystem, which is configured to route the requests to members of a DB2 datasharing group on different LPARs.

This connection will be balanced through a DVIPA and the Sysplex Distributor.

The price that is paid for the additional workload balancing is the remote connection overhead between the DB2 subsystems.

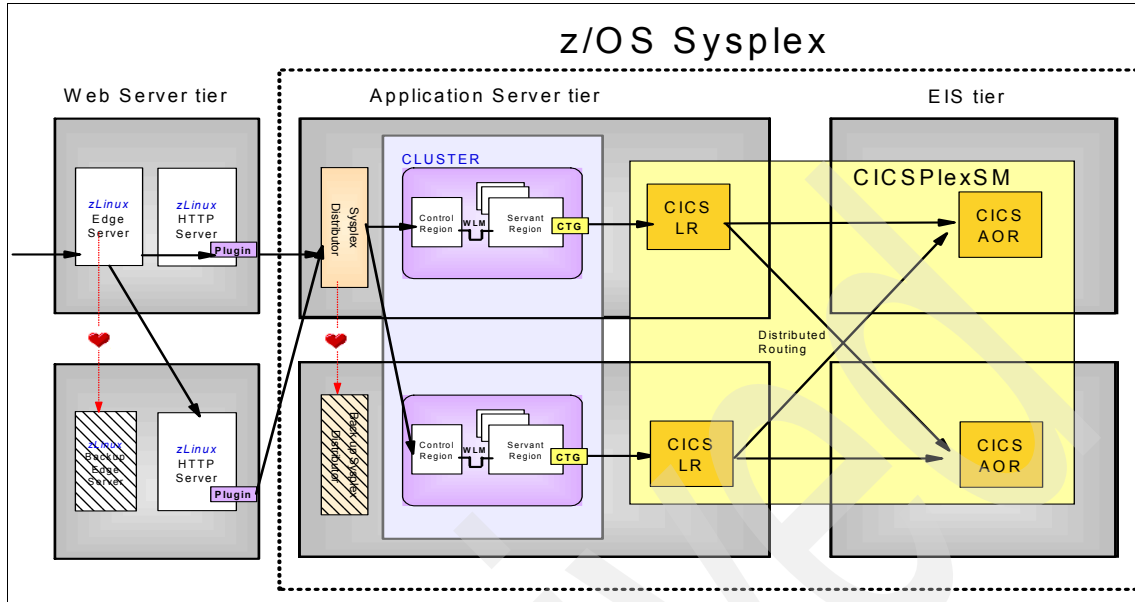


Figure 5-9 Three-tier with failover support for CICS

Figure 5-9 shows a high-availability solution for accessing CICS in a three-tier solution where the EIS tier is remote to the EJB tier.

This scenario shows the CTG connecting locally to a CICS Listening Region on the same LPAR. The application in the local CICS region can make use of CICS Distributed Routing to forward the request to an AOR in the CICSPlex SM configuration on the same LPAR, or as in this scenario on a different LPAR.

If one of the AORs goes down, then the request is routed to another available AOR in the group.

5.7.2 Decouple Web and EJB layers

In this scenario, WebSphere Application Servers are configured as “appliance” servers, running only a specific component of the whole application. The Web tier will be comprised of application servers that contain Web applications, and the EJB tier will comprise application servers that only run EJB and Java Connector Architecture (J2C) applications.

Using this approach will allow the servers to be configured to specialize either on the Web or EJB container. The EJB containers are on the same LPARs as the EIS subsystems so they can benefit from the performance and security advantages provided with local connections.

This three-tier infrastructure would be well-suited to environments where z/OS is to be used mainly for business logic and connection to resources. The Web applications could be hosted on a different platform. There is a performance impact in that all calls between the Web and EJB containers must be remote. This precludes the use of the local interface or any “prefer local” processing between the Web and EJB layers. All calls between the Web and EJB containers will be remote I/O requests that require marshalling and demarshalling.

This overhead can be controlled by the use of coarse-grained remote calls to the EJBs using a Data Transfer Object (DTO) as described in “Data Transfer Object” on page 48. The overhead of serialization should be considered as trade-off between the use of “appliance” servers for high CPU/High memory activities and the overhead of remote network calls. The serialization overhead can be further reduced by ensuring the DTO is not too complex, in particular that it does not contain deeply nested objects. The complexity of the object has a direct impact on the serialization and marshalling overhead.

As mentioned above, this scenario is more suited where the Web or presentation tier is placed on a different platform. A large amount of Web processing is non-file-based and significantly memory bound. With caching configurations such as WebSphere Dynacache this becomes even more memory-based.

The separation of the platforms allows the z/OS platform to be focused on processing business transactions and other potentially cheaper platforms such as Linux to be used to process user interface activity.

A correctly designed Web application will have caching enabled and so only a limited number of business-centric requests will be processed on the z/OS platform running the EJB and EIS.

The presentation tier can then be scaled and cheaper hardware platforms used. The use of Linux presents an interesting situation, that is, to move the Linux platform process—the presentation tier—back up to Linux on zSeries. This decision should be based on the expected capacity and the availability of MIPS, and particularly memory.

The separation of presentation and business tiers allows the tuning and resource usage of the underlying hardware to be focused onto a particular type of activity.

Availability

As stated previously, the three-tier architecture is no different from the two-tier architecture when trying to create a highly available solution. Redundancy and failover support are the keys to availability.

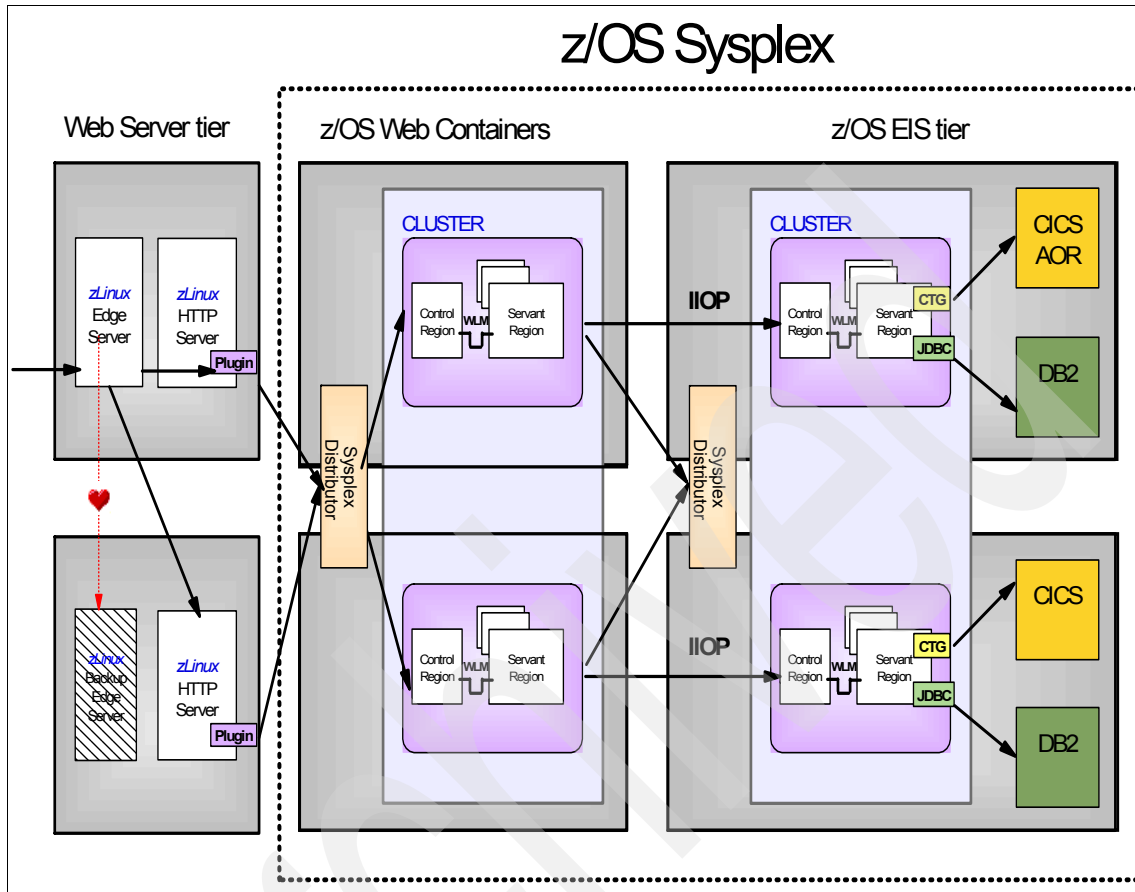


Figure 5-10 Three-tier separate Web and EJB layers

Providing redundancy is more complicated in that we are now dealing with two distinct types of application server. In Figure 5-10 both the Web and EJB tiers have two application servers spread over two nodes on different physical processors to provide redundancy and failover.

The initial availability mechanism is provided by having multiple Edge Servers (Network Dispatchers) distributing workload to the HTTP servers. The Edge Server can provide a single point of failure and so a primary and backup Edge Server should be used. These edge servers offer a cluster address to the user and the Edge Server is the single point of entry into the system.

The Edge Servers distribute workload across multiple HTTP servers, avoiding a single point of failure. The WebSphere plug-in on each of the HTTP servers is then responsible for distributing the workload across multiple Web containers.

Session affinity is provided through the HTTP server plug-in that redirects users back to the Web container that contains the existing session through the use of the JSESSIONID cookie. Non-affinity-based requests are directed either through a DVIPA for native z/OS-based implementation, or in a random or round-robin fashion to the Web containers.

IIO requests from the Web containers to the EJB containers are then directed to the application servers on the other LPARs. The workload management of IIO requests coming from an EJB client is described in “IIO workflow in a Parallel Sysplex” on page 64. In this case the EJB client is the Web application running in a separate container.

Interoperability

By separating the Web and EJB tiers we have introduced the need for an interoperable situation between the ORBs in the Web tier and those on the EJB tier. This needs to be kept in mind when maintaining each tier. Although IBM tests many interoperability scenarios, the WebSphere Application Server versions on each tier should be kept as close as possible to minimize any risk of incompatibility. Maintaining this interoperability dependency can be more difficult if the Web tier is on a different operating system platform from the EJB tier since the maintenance policies are likely to be very different between the two.

Decouple namespace lookups

When an application on the Web tier needs to use an EJB component on the EJB tier, it must first look up the component’s home in the namespace of the EJB container.

In this situation the Web component is the client to the EJB that is being used. A name service is used to carry out the retrieval of the IOR for the home object from the namespace.

The deployment manager, node agents, and application servers all provide a name service that the client can bootstrap into and use. The most highly available option is to bootstrap into the name service for the node agents in a cell listening on the same DVIPA address. If one of these goes down, the others are still available to service the request.

Another issue to consider is the binding between the EJB reference on the calling component on the Web tier and the actual JNDI name of the component that is being looked up on the EJB tier. This binding is made during application deployment. The problem is that this binding refers to the application server name or cluster name in the JNDI string.

The problem is that the reference is now tied to the target topology—the server name or cluster name. If the target EJB needs to be redeployed to another server for, say, operational or capacity planning reasons, then all clients using that EJB

need to be identified and have their reference bindings reworked. This is somewhat of a restriction to the flexibility that decoupling the tiers should allow.

Therefore, IBM introduced Configured Bindings with WebSphere Application Server V5. An EJB Configured Binding provides indirection for the client's lookup and is basically an alias to the actual EJB JNDI name. The client will look up the JNDI name of the configured binding, which itself points to the actual EJB JNDI name. The clients are no longer concerned with where the actual EJB is running or registered, so any changes in the topology can be hidden from them.

Security

The implementation of separate Web and EJB/EIS layers provides some of the same challenges that exist for the decoupling of the Web and EJB from the EIS tier. WebSphere Application Server, however, does allow for some variations in connections security.

The fundamental difference in this configuration is that the layer separation is between application servers, that is, the Web container and EJB container. WebSphere Application Server provides IIOP-based connectivity using various ports providing different levels of security between different instances running on the same or different platforms.

The security challenge is two-fold. The first issue is to transfer the identity of the user from the calling to the called instance of WebSphere Application Server; the second is doing that over a secure line. All other issues are covered according to features and functions for all application server instances.

The system can be configured for the outbound IIOP connection to support basic authentication, client certificate, or identity assertion over SSL. This allows an identity to be passed from server to downstream server, in our case Web container to EJB container, in a secure manner. The downstream server must also be configured to accept and recognize this identity.

The establishment of an SSL connection can either be done using client or mutual authentication.

This identity can be in the form of a username and password, a client certificate, or an identity token. The use of identity assertion is possible where the target EJB is called with RunAs=caller specified. However, care must be taken to ensure that the identity management subsystem supporting both Web and EJB container is the same. In short, the z/OS identity management system must contain details of all users that will be providing requests from the Web container even if the Web container is not on the z/OS platform.



WebSphere platform messaging

In this chapter we discuss the high-availability and workload management considerations that must be taken into account when asynchronous messaging is used in the WebSphere Application Server for z/OS environment.

If you do not plan to use messaging, then this chapter does not apply to your environment.

6.1 Overview

The WebSphere Application Server for z/OS supports the following JMS providers:

- ▶ Default messaging
- ▶ WebSphere MQ
- ▶ Generic
- ▶ V5 default messaging

The sections that follow consider only the default messaging provider and how it can be used to achieve high availability and workload management.

For information on using WebSphere MQ in a highly available environment, see *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864.

6.2 Concepts and architecture

The service integration bus provides the underlying technology for the default messaging provider. The service integration bus introduces a number of new concepts that are discussed in the following sections. The discussion will also highlight any details that are unique to the WebSphere Application Server for z/OS.

Note: For further information on the service integration bus, refer to Chapter 10, “Asynchronous messaging” and Chapter 11, “Default messaging provider” in the book, *WebSphere Application Server V6: System Management & Configuration Handbook*, SG24-6451

6.2.1 Buses

The default messaging provider introduces a new administrative concept referred to as the *bus*. It allows the administrator to group together the messaging resources that make up the bus into a single entity, without having to expose all of the details to applications connecting to the bus.

6.2.2 Bus members

There are two types of bus members: a stand-alone application server and a cluster of application servers. An administrator can add instances of either type to a bus resulting in that instance becoming a member of the bus. The scope of a

bus is limited to the cell. This means that only application servers and clusters defined in the same cell as the bus can become members of that bus.

Becoming a bus member automatically enables a number of messaging resources on the application server or cluster. In the WebSphere Application Server for z/OS, this includes activating the Control Region Adjunct (CRA) in every stand-alone application server, or cluster member. The CRA acts as the container for any messaging engines that are created.

From a high availability and workload balancing perspective we will mainly be interested in cluster bus members.

6.2.3 Messaging engines

A *messaging engine* is the component inside an application server that provides the core messaging functionality. When a stand-alone application server is added as a member of a bus, a single messaging engine is automatically created and associated with the application server. In the WebSphere Application Server for z/OS, this messaging engine will be hosted in the CRA. This is shown in Figure 6-1.

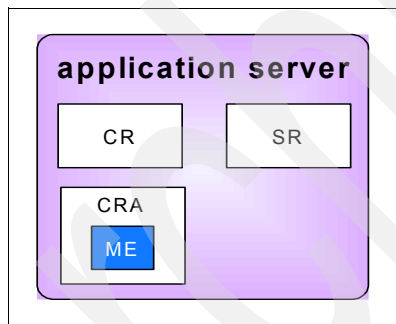


Figure 6-1 A single messaging engine added to an application server

When a cluster of application servers is added as a member of a bus, a single messaging engine is automatically created and associated with each cluster member.

This means that the *same* messaging engine is created at each cluster member, however, only one of these instances can be active at any one time. The others act as standby instances that can potentially take over if the active messaging engine fails.

This process is referred to as *failover*, and is discussed in 6.3.1, “Failover” on page 130.

Figure 6-2 shows a cluster containing the same messaging engine in every server in the cluster, however, only the messaging engine in server A is active. The messaging engines in server B and server C are available as standbys in case the messaging engine in server A fails.

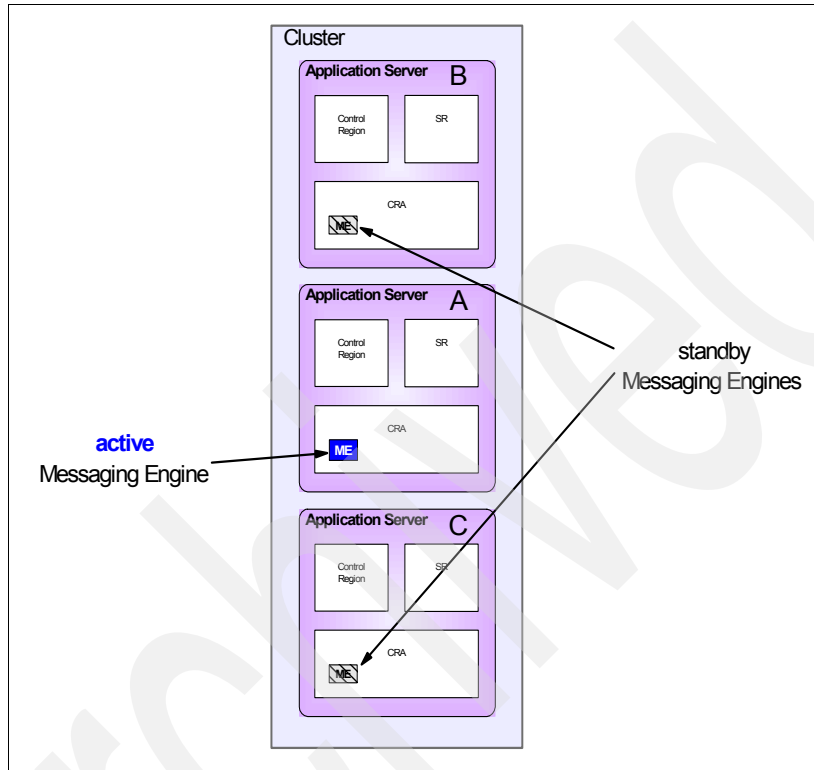


Figure 6-2 A single messaging engine added to a cluster

An administrator has the ability to add additional messaging engines to a bus member. In the WebSphere Application Server for z/OS, all messaging engines that are associated with an application server or cluster member are hosted in the single CRA belonging to that server.

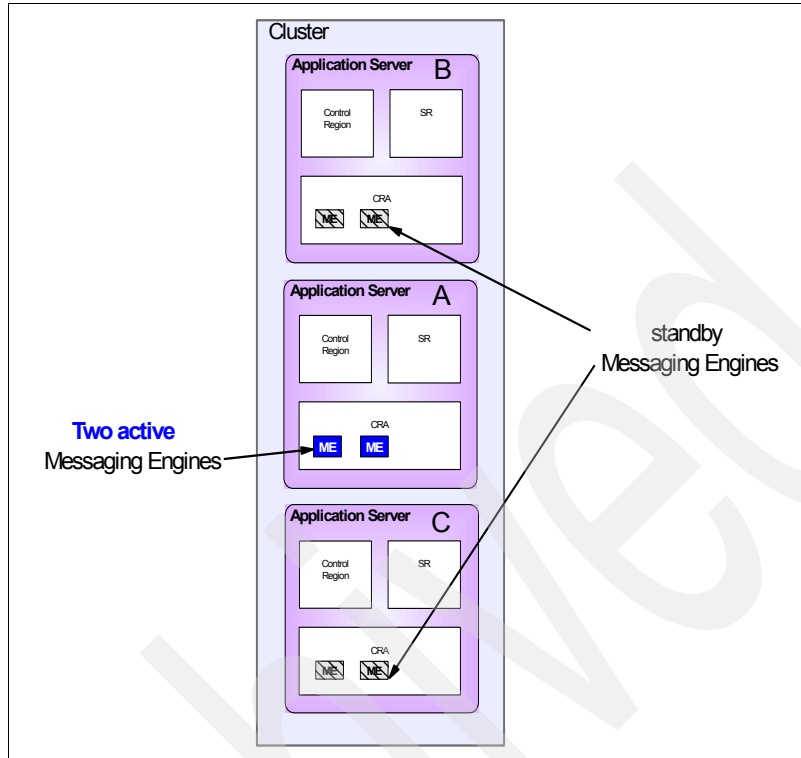


Figure 6-3 Two messaging engines added to a cluster

In Figure 6-3 we can see that a second messaging engine has been added to the cluster. In the diagram, both messaging engines are active on server A.

By default, all messaging engines will become active on the first server in the cluster to become available. For the details on how to modify this behavior see 6.6, “Core group policies” on page 145.

6.2.4 Data stores

Every messaging engine in a bus has its own *data store*. The data store consists of a set of tables in a relational database, such as DB2 Universal Database™ for z/OS. These tables are all stored under a unique schema. The messaging engine uses the data store to hold durable data such as persistent messages.

Figure 6-4 on page 128 shows the active messaging engine on server A connected to a relational database. The standby messaging engines will not attempt to connect to the database until either of them is activated. However for

failover to occur successfully, the data store must be accessible from every server in the cluster.

A messaging engine holds a table lock on one of the tables in the data store. This is to prevent data corruption due to concurrent access to the data held in the tables. In a cluster bus member, the active messaging engine holds the table lock. If it fails then the table lock must be released before failover can take place.

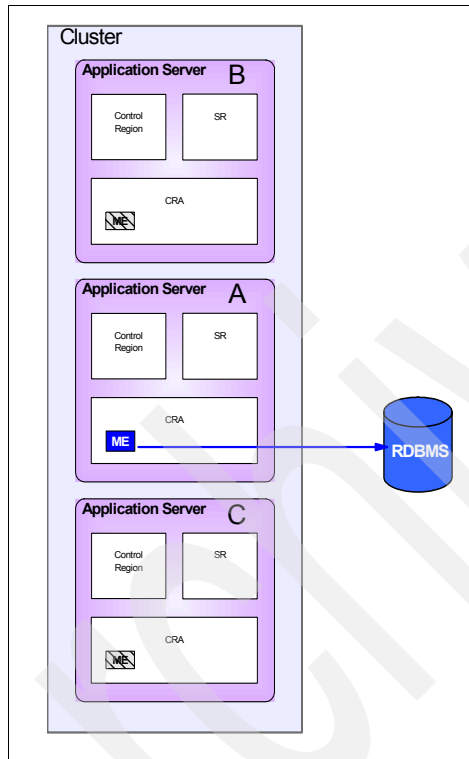


Figure 6-4 Messaging engine connected to a RDBMS

6.2.5 Destinations

Destinations are logical addresses within a bus to which applications can attach as message producers, or message consumers. There are four different types of destinations:

- ▶ Queue destinations

A queue is a destination which is configured for point-to-point messaging. While a queue is a bus wide resource, a queue will have a *queue point*,

created when the queue is defined. This queue point is *localized* to a single bus member.

If the bus member is a stand-alone application server, the queue point is localized to the messaging engine on that application server. If the bus member is a cluster, the queue point is localized to every messaging engine defined within the cluster. Each of the queue points in the cluster is referred to as a *partition* of the destination.

If an application sends a message to a queue destination that has been localized to a cluster then that message will only be delivered to one partition of the destination. If another application tries to consume messages from the same destination then it will only be able to receive messages from one of the partitions, even if there are messages available in other partitions.

- ▶ Topic space destinations

A topic space is a destination which can be used for publish/subscribe messaging. A topic space is also a bus wide resource but, unlike a queue, the *publication point* will automatically be localized to every messaging engine in the bus.

- ▶ Alias destinations

An alias destination is a destination that can be used to refer to another destination.

- ▶ Foreign destinations

A foreign destination is a destination that can be used to refer to a destination on another bus.

6.2.6 Mediations

A *mediation* is used to process messages flowing through the bus. The types of processing that can be done by a mediation include:

- ▶ Transforming a message from one format to another.
- ▶ Routing messages to destinations that were not specified by the target destination.
- ▶ Augmenting messages by adding data from a data source.
- ▶ Discarding messages.

A mediation is associated with a destination on a bus. A corresponding *mediation point* is automatically created and associated with the destination. This means that any messages that arrive at the destination will be processed by the mediation.

6.2.7 Foreign buses

A service integration bus can be configured to connect to, and exchange messages with, other messaging networks. In order to do this, a *foreign bus* must be configured.

A foreign bus may represent:

- ▶ A service integration bus in the same WebSphere Application Server V6 cell as the local bus.
- ▶ A service integration bus in a different WebSphere Application Server V6 cell from the local bus.
- ▶ A WebSphere MQ network.

The impact of foreign buses on availability is discussed in “Foreign buses and failover” on page 133.

6.3 Availability

The service integration bus (SIB) supports two options for achieving high availability: failover and messaging engine clustering.

6.3.1 Failover

If the active messaging engine in a cluster fails, the high availability management component of the WebSphere Application Server for z/OS selects one of the standby instances to become active. Ideally, the active and standby instances run on different zSeries servers so that even if there is a hardware failure, the messaging engine can recover in a timely manner.

In the WebSphere Application Server for z/OS, if a message engine fails then this may result in the CRA being terminated. The server as a whole, however, should survive such a failure. If the CRA terminates it will be restarted by the CR.

We can see this process taking place in Figure 6-5. The messaging engine in server A fails, and brings down the CRA. The standby messaging engine, in Server B, takes over for the failed instance. Once the CRA in server A is restarted the messaging engine in that server is made a standby instance.

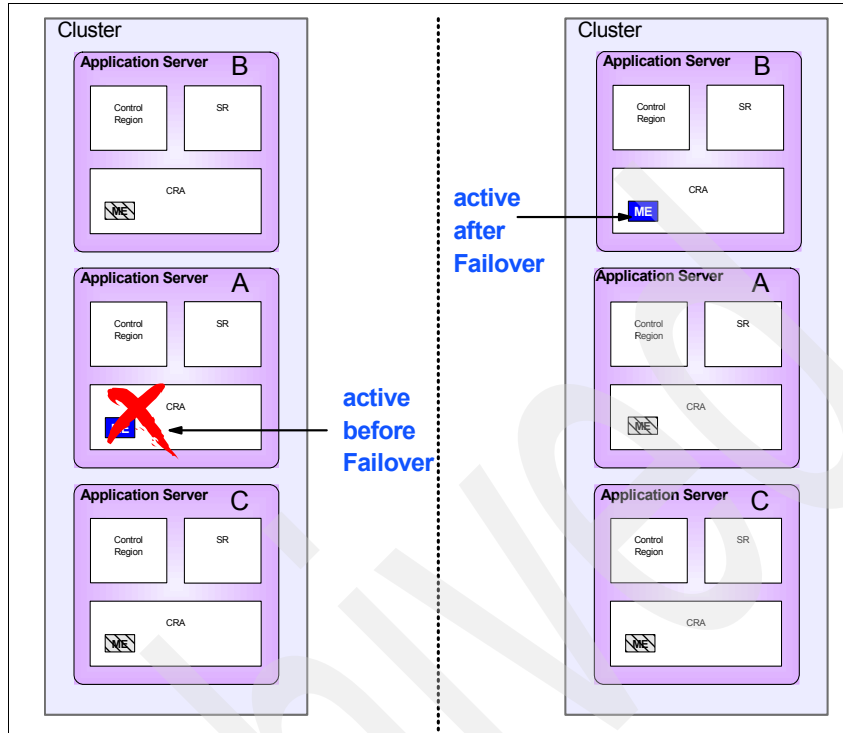


Figure 6-5 Messaging engine failover

It is possible to control which servers a messaging engine is allowed to run on, and in effect which servers a messaging engine can failover to, using policies. For further details, see 6.6, “Core group policies” on page 145.

Message reliability

Whether the messages, stored on a partition that is managed by the messaging engine that has failed, will still be available to a consuming application once the messaging engine has completed failover will depend on the reliability level of each message.

The SIB (service integration bus) supports five reliability levels as shown in Table 6-1 on page 132.

Table 6-1 Reliability levels

Reliability	Description
Best Effort nonpersistent	Messages are discarded when a messaging engine is stopped, or if it fails. Messages may also be discarded if the connection used to send them becomes unavailable or as a result of constrained system resources.
Express nonpersistent	Messages are discarded when a messaging engine is stopped, or if it fails. Messages may also be discarded if the connection used to send them becomes unavailable
Reliable nonpersistent	Messages are discarded when a messaging engine is stopped, or if it fails.
Reliable persistent	Messages are discarded when a messaging engine fails, but not when it is stopped normally.
Assured persistent	Messages are never discarded.

As message reliability increases, the corresponding message throughput and performance will decrease. The JMS connection factory, available from the default messaging provider, allows us to map these reliability levels to the corresponding JMS delivery modes PERSISTENT/NON_PERSISTENT.

Transactions

If a messaging engine fails, any *in-flight* transactions, transactions that have not yet been committed, associated with that messaging engine will be rolled back. This means that any messages that were put to the destination will be removed, and any messages that were retrieved from the destination will be put back.

Any *in-doubt* transactions, prepared transactions for which we do not know whether the final outcome is commit or rollback, will remain in-doubt until they are resolved by the transaction manager. This means that any messages associated with these transactions will remain unavailable until the transactions are resolved.

Message availability

If a messaging engine, containing a partition of a destination, fails then no new messages will be sent to that partition. If there are any messages already held at that partition, however, then those messages will either be *discarded*, or become temporarily *unavailable*.

Messages will only be discarded if it is allowed by the reliability level of the messages, or if the messages have expired. The remaining messages on the partition will be unavailable until the messaging engine has completed failover, and applications are once again allowed to consume messages from that partition.

In a sense, the availability characteristics of the partition, match the availability characteristics of the messaging engine that hosts that partition.

Foreign buses and failover

When a messaging engine is connected to a WebSphere MQ queue manager there are additional high availability considerations to take into account. If this messaging engine fails over to another server, then it is possible that the link to the WebSphere MQ queue manager can be broken. For further details see the following InfoCenter article:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc\concepts\cjt0003_.html

6.3.2 Messaging engine clustering

Failover can significantly improve the availability of applications using asynchronous messaging. Application availability can be improved even further, however, by adding additional messaging engines to the cluster.

Figure 6-6 shows a cluster that has been configured to have an active messaging engine in every server. Inbound work sent to a destination in that cluster is evenly distributed among all of three messaging engines. If the messaging engine in server **A** fails then that messaging engine is no longer available to process new incoming work. However, this work can be redistributed among the remaining messaging engines in servers **B**, and **C**.

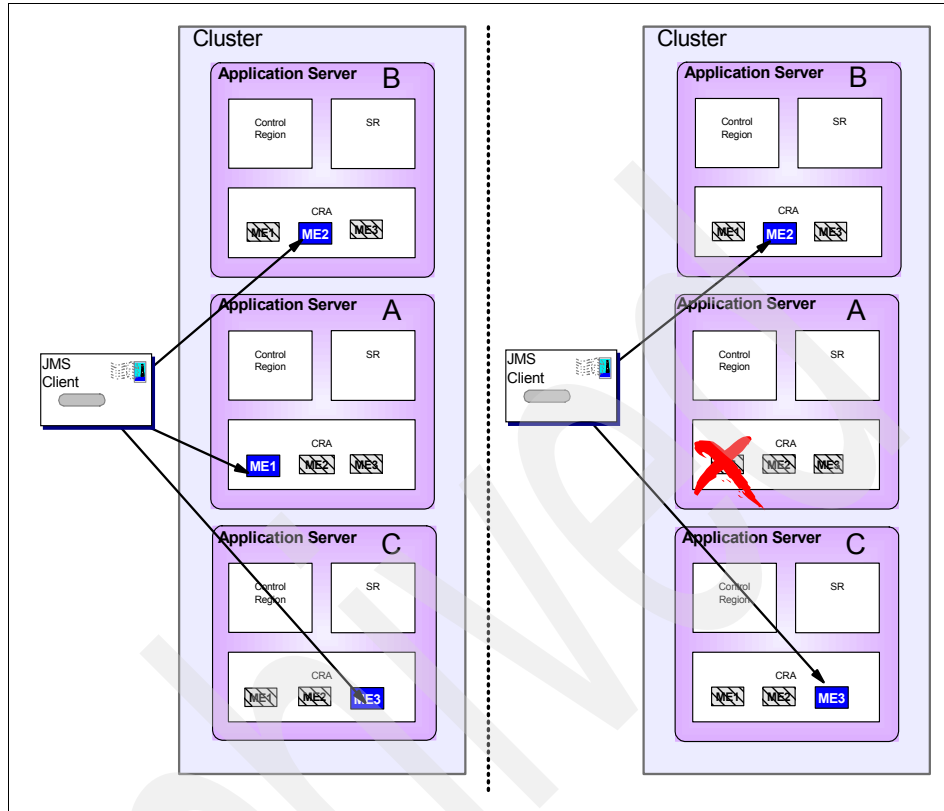


Figure 6-6 Messaging engine clustering

The details of how messages are distributed across the messaging engines in a cluster are discussed in 6.5, “Workload balancing” on page 135.

6.4 Scalability

Adding additional messaging engines to a cluster provides scalability as well as availability. As described in 6.3.2, “Messaging engine clustering” on page 133, additional messaging engines can be added to a cluster bus member. When this is done, inbound work is redistributed to include both the existing messaging engines, and any new messaging engines. This is shown in Figure 6-7 on page 135.

In the diagram we see work being sent to a destination localized in the cluster. Initially, there are two messaging engines, active on server A, and server B. Each messaging engine will have a partition of the destination.

To improve the message throughput a third messaging engine can be added to server C. This new messaging engine will also have a partition of the destination. The incoming work can now be redistributed among the three messaging engines.

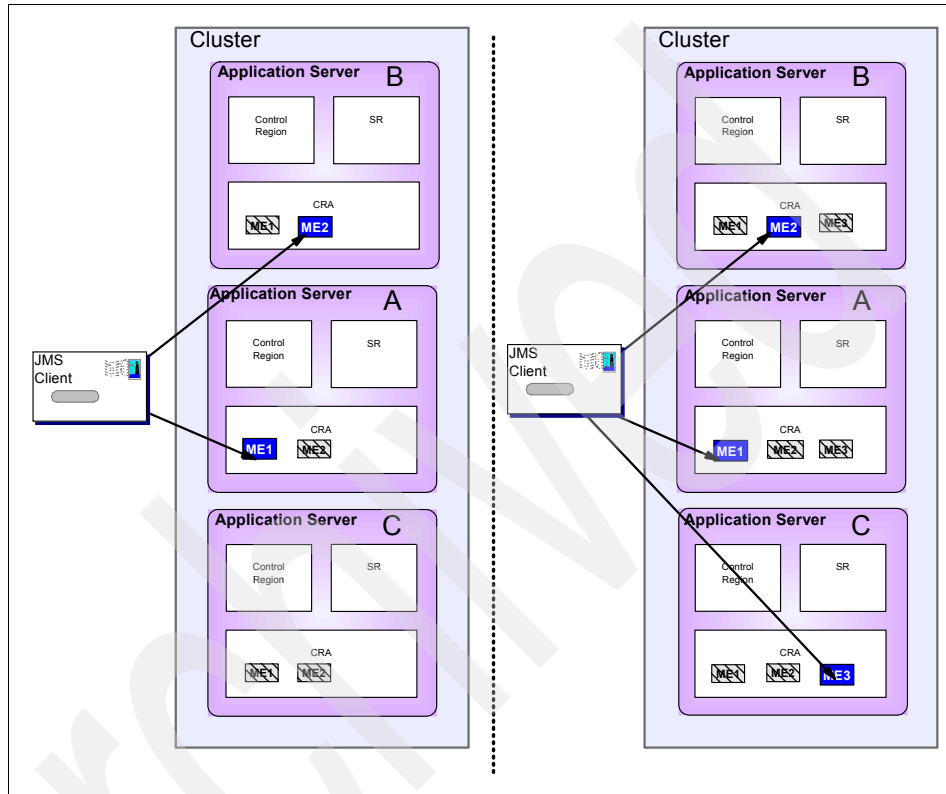


Figure 6-7 Messaging engine scalability

6.5 Workload balancing

The SIB (service integration bus) supports two styles of distributing work between the messaging engines in a cluster. These are *push* workload balancing, and *pull* workload balancing. Before we discuss these styles, however, we first need to understand how a client can connect into a SIB, as this has a significant impact on the ability to balance the workload.

6.5.1 Connecting to a service integration bus

A JMS client obtains connections to a SIB (service integration bus) using a JMS connection factory, defined for the default messaging provider.

The messaging engine within a particular bus that a JMS client will actually connect to, depends on the connection properties defined within the JMS connection factory. Before those properties can be applied to selecting a messaging engine, however, there must first be a way to discover what messaging engines are available within the bus.

There are two scenarios to consider:

- ▶ Clients running inside of WebSphere Application Server

In this case, the connection factory is automatically able to communicate with other components of the WebSphere Application Server runtime in order to determine what messaging engines are available inside the bus, and where they are located.

- ▶ Clients running outside of WebSphere Application Server

In this case, the connection factory does not have access to the information about the messaging engines inside the bus. This may be because the client is executing outside of the WebSphere Application Server environment, or in a WebSphere Application Server environment in a different cell to the target bus.

In order to obtain this information, the connection factory must connect to an application server within the same cell as the bus. This server is called a *bootstrap server*.

A bootstrap server is simply an application server that is running the SIB service. In the WebSphere Application Server for z/OS, this means that the CRA is active, but it does not necessarily host any messaging engines. The bootstrap servers that a connection factory can connect to are configured using the *provider endpoints* property of the connection factory.

Messaging engine selection

The most important property on the JMS connection factory, of the default messaging provider, is the bus name. In the absence of any other connection properties, the connection factory is able to return a connection to any available messaging engine in the bus.

However, there are a few simple rules that are applied in order to determine the most suitable messaging engine:

1. The connection factory looks for a messaging engine, within the specified bus, that is in the same server as the JMS client. If such a messaging engine

is available then a connection is made between the JMS client and the messaging engine.

2. If it was not possible for the connection factory to create a connection to a messaging engine in the same server as the JMS client, the connection factory will look for a messaging engine in a server on the same host. If multiple messaging engines are available on the same host as the JMS client, new connections to the target bus will be workload balanced across them.
3. If it was not possible for the connection factory to create a connection to a messaging engine on the same host as the JMS client, the connection factory will look for any other messaging engine that is in the specified bus. If multiple messaging engines are available within the target bus, new connections to the bus will be workload balanced across them.
4. If it was not possible for the connection factory to create a connection to a messaging engine in the specified bus, then a JMS exception is returned to the client.

For details on all the JMS connection factory properties see Chapter 10, “Asynchronous messaging” in *WebSphere Application Server V6: System Management & Configuration Handbook*, SG24-6451.

Message-driven beans

We have covered the rules that determine how JMS clients connect to a messaging engine, but MDBs are a special case.

It is the responsibility of the SIB JMS Resource Adapter to connect to a bus, not a Message Driven Bean (MDB), in order to consume messages from a destination.

Note: The MDB is still responsible for connecting to the bus in order to produce messages.

When a message arrives at that destination, the resource adapter (RA) will invoke an instance of the MDB to consume the message.

The resource adapter (RA) obeys a slightly different set of rules for connecting to a messaging engine in a bus.

1. Only attempt to connect to a messaging engine in the same server as the MDB, if one is defined there, even if it is not currently active.
2. If there is no messaging engine in the same server as the MDB then follow the rules 1-4 as documented in “Messaging engine selection” on page 136.

The consequence of this difference can be seen in Figure 6-8. The diagram shows an MDB deployed to a cluster, that is a member of a bus. There is only one messaging engine in the cluster, and it is active on server **A**.

In this configuration, messages arriving at server **A** will only be able to drive the MDB in server **A**. The MDBs in the other servers will not be able to consume any of the messages that arrive at server **A**.

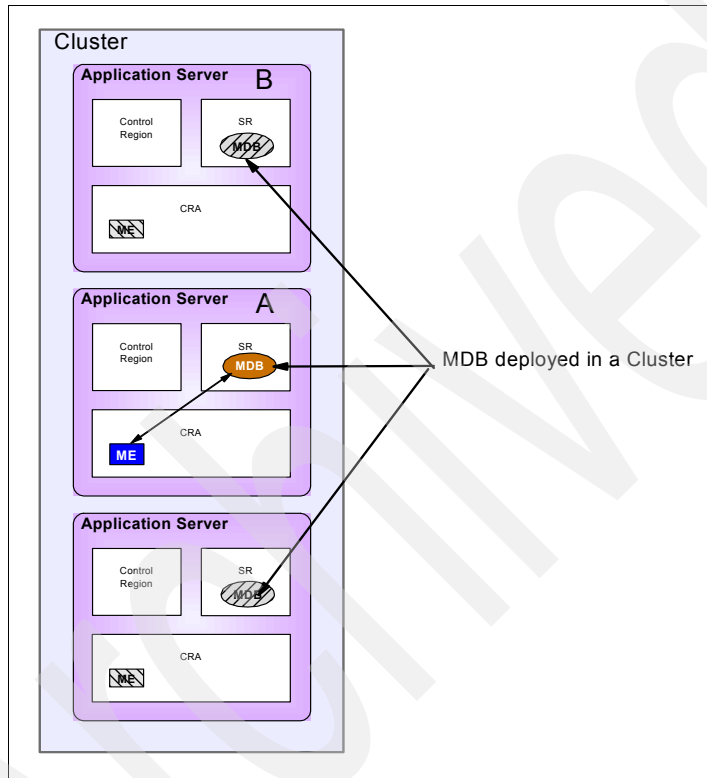


Figure 6-8 MDB application in a cluster with one messaging engine

Should the active messaging engine in server A fail, then the messaging engine will failover to another server in the cluster. The MDB in that server will be connected to the messaging engine and start receiving messages.

It is also possible to configure the cluster shown in Figure 6-8 such that there is an active messaging engine in every server. In this case, MDBs in every server would be able to consume messages, but only from the partition that is local to the server containing the MDB. Essentially, the MDBs reflect the clustered nature of the underlying messaging engines.

An MDB, installed on a server in the same bus as the cluster, can also consume messages from the destination localized to the cluster. An MDB, installed in a server outside of the bus, can also consume messages from a destination that is localized to cluster that is a member of a bus. These are shown in Figure 6-9.

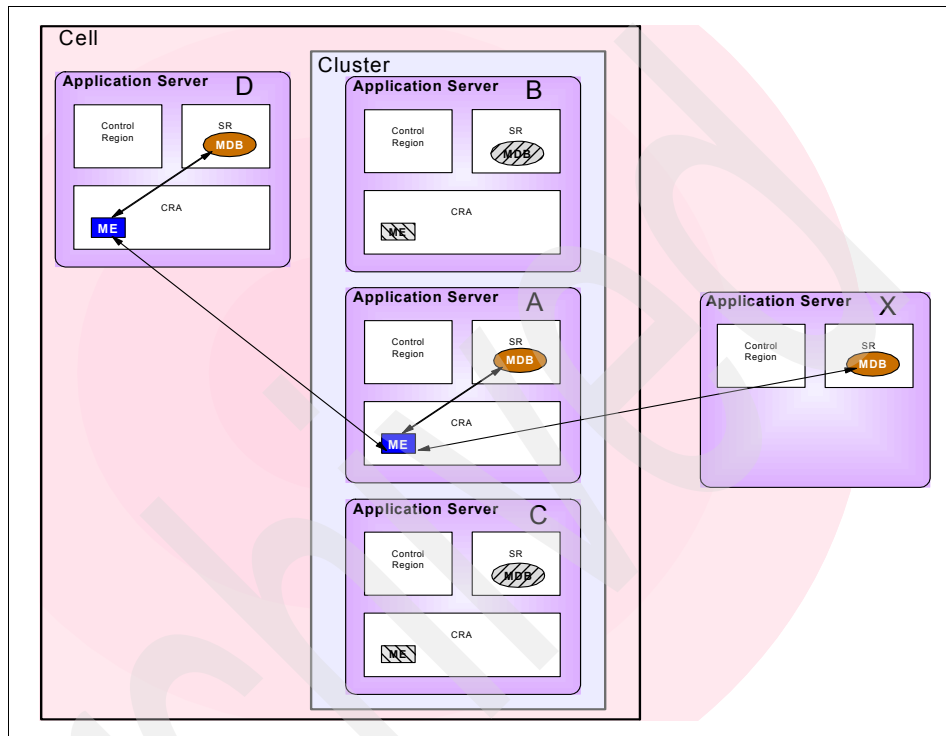


Figure 6-9 MDB application in servers outside of the cluster

Figure 6-9 shows MDBs running in server **D** and server **X**. Server **D** is inside the bus, and has one messaging engine. Server **X** does not have any messaging engine as it is outside of the bus. The MDB in server **D** will connect to the local messaging engine, in order to consume messages from the destination localized to its server.

The resource adapters running in servers **D** and **X** connect to the messaging engine running in server **A** in order to consume messages from the partition destination on that server.

If there is more than one active messaging engines in a single server of the cluster, as may occur due to failover, then the MDB in that server will consume messages from all of the partitions associated with those messaging engines.

6.5.2 Push workload balancing

In order to achieve push workload balancing we require the following:

- ▶ Message producers must ensure that the messages they produce will be workload managed onto different partitions of a queue-type destination. This can be done in a number of ways, including:
 - JMS clients connecting directly into the cluster should be workload balanced among the messaging engines available in the cluster.
 - Clients connecting to messaging engines that are not in the server cluster. Once a messaging engine that is outside of the cluster accepts a message it becomes responsible for routing the message through the bus to a partition for the destination.
 - Messages produced by an EJB/servlet installed in the cluster. As the messages produced by the EJB/servlet will either be sent to the local partition, if there is a local one, or to one of the partitions in the other servers, if there isn't a local one, the messages will effectively be workload balanced across the available partitions.
- ▶ Message consumers must connect to each partition of the destination in order to consume messages. If any partitions do not have consumers then any messages sent to that partition may never be consumed.
 - The best way to ensure that every partition has a consumer is to install an MDB in the server cluster, that is configured to consume from that destination.

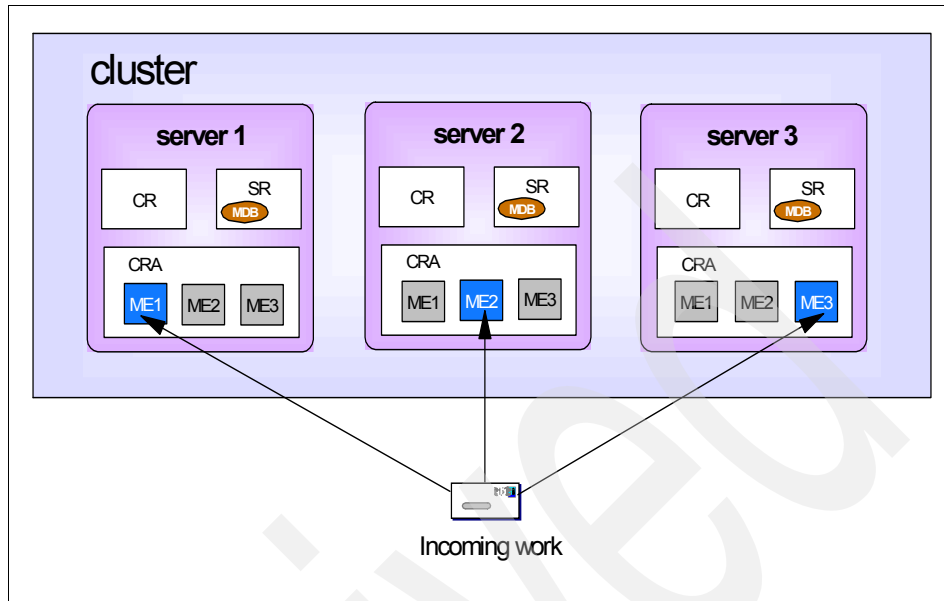


Figure 6-10 Messages are distributed between the active messaging engines

Figure 6-10 shows a cluster that is configured such that there is an active messaging engine in every member of the cluster. This means that any destination that has been localized to the cluster will be partitioned across all of the messaging engines (the partitions are not shown in the diagram).

Each MDB running in the cluster will consume the messages that are delivered to the local partition. In this way, the message processing workload can be evenly distributed among all of the servers.

Workload balancing bootstrapped clients

As described in 6.5.1, “Connecting to a service integration bus” on page 136, clients that want to connect to a service integration bus from outside of the WebSphere Application Server environment need to use a bootstrap server. There is an unfortunate side-effect of the selection rules when the provider endpoints are the servers in the cluster.

If there are many JMS clients using the same connection factory, they will all bootstrap using the same list of bootstrap servers. Because the connection factory will attempt to connect to a bootstrap server in the order in which they are specified in the provider endpoints list, it is likely that all of the JMS clients will be connected to the same messaging engine in the first available bootstrap server. The JMS clients will not be load balanced across the messaging engines in the cluster.

There are a couple of solutions to this problem:

- ▶ Use a dedicated bootstrap server that is not running a messaging engine for the target bus. This will ensure that the connections established for JMS clients are load balanced across the available messaging engines in the bus.
- ▶ Provide different clients with differently configured connection factories, each of which has a different provider endpoint in the first position on the list.

We expect this problem to be addressed in a future release of WebSphere Application Server for z/OS.

6.5.3 Pull workload balancing

In order to achieve pull workload balancing we require the following:

- ▶ A destination with only a single partition. This is to ensure that all messages that arrive at that destination will be consumed.

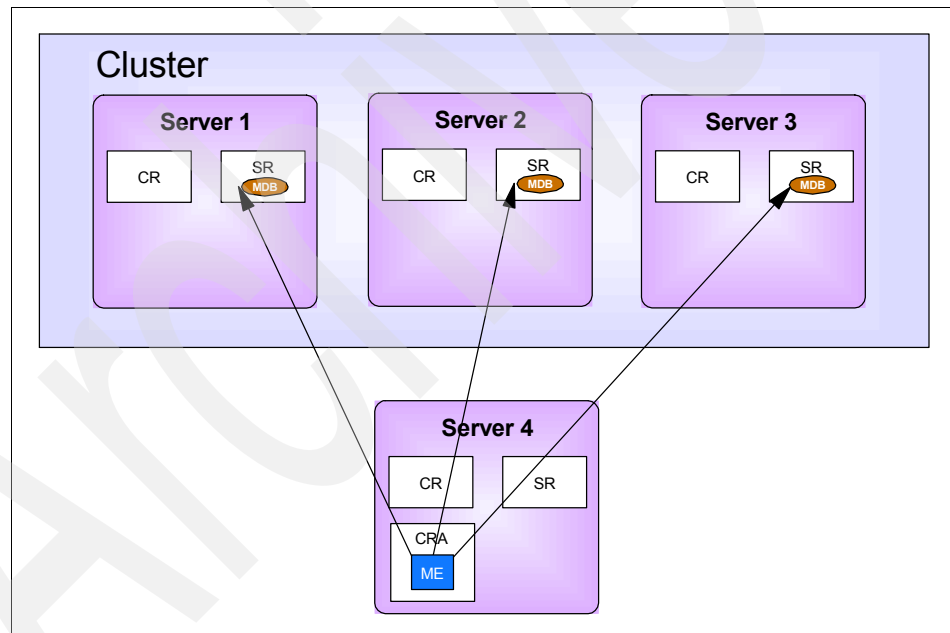


Figure 6-11 MDBs in the cluster consume from the same destination

In Figure 6-11 we see that Server 4 has one messaging engine. An MDB application is installed in the cluster, which is configured to consume messages from a destination (not shown in the diagram) localized to Server 4.

In this configuration, the resource adapter in each server will connect independently to the messaging engine in Server 4, in order to consume messages from that destination. In this way, the workload for consuming and processing the messages will be distributed across the cluster.

There will be no interruption of service if one of the servers in the cluster fails, as the MDBs in the other servers can continue to consume messages. If Server 4 were to fail, however, then no work could be done until it is restarted.

In order to improve the availability, we can configure the system as shown in Figure 6-12. Here we have a new server, Server 5, that has been clustered with Server 4. If the messaging engine in Server 4 were to fail then it can failover to Server 5. This will allow the MDBs running in cluster 1 to continue to consume messages while the problems with Server 4 are rectified.

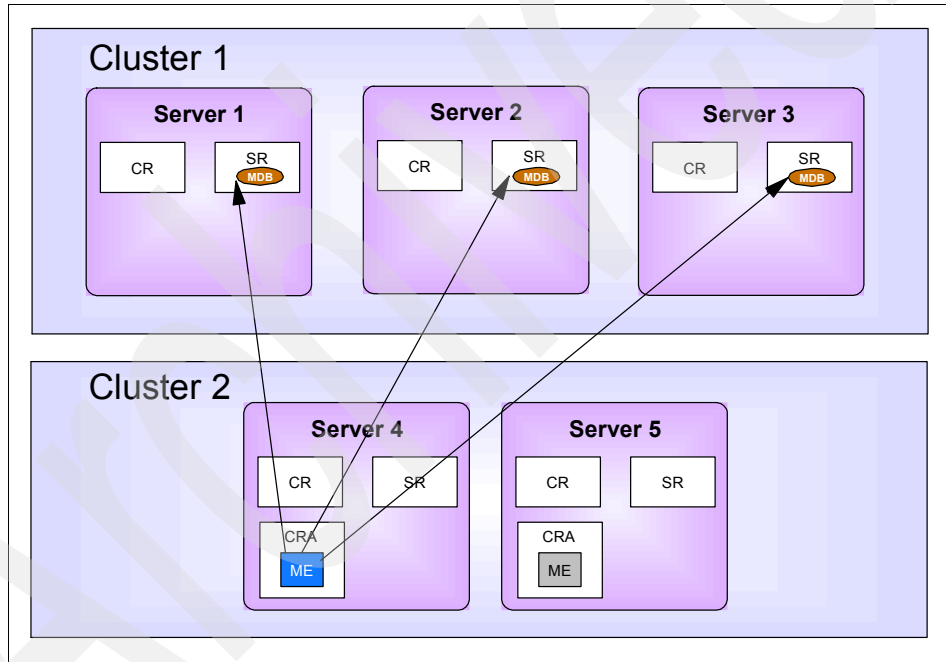


Figure 6-12 A second cluster improves availability

Note that we do not add additional messaging engines to Cluster 2. This would increase the number of partitions of the destination, and then we could not guarantee that all of the partitions would have consumers.

Publish/subscribe and workload balancing

Pull workload balancing will also work in a publish/subscribe scenario. In nondurable publish/subscribe messaging, all subscribers typically receive a copy of each matching publication. In order to spread the workload across a cluster of subscribers, however, the incoming publications must be divided between the subscribers, instead of every subscriber receiving every publication.

If the subscribers are MDBs in a cluster, such as the configuration in Figure 6-12 on page 143, they are likely to be using the same connection factory, so they will have the same client identifier. In general, two subscribers trying to use the same client identifier and subscription name would be treated as an error, but for durable subscriptions, you can control whether multiple subscribers with the same client identifier are treated as instances of the "same" consumer.

This is controlled using the Share durable subscriptions attribute of the connection factory. A shared durable subscription is one which can be simultaneously accessed by multiple cooperating clients, such as MDBs. Because the subscribers are treated as cooperating rather than contending, each message is delivered to one MDB.

6.5.4 WLM Classifier

In WebSphere Application Server for z/OS, you can use z/OS transaction classes to classify client workload for workload management. If the work has a high priority, workload management can direct the work to a high-priority servant in the server. If the work has a low priority, workload management can direct the work to a low-priority servant. The effect is to partition the work according to priority within the same server.

This facility is also supported by the service integration bus. An administrator has the ability to define rules for classifying the messages that flow through a messaging engine. The messaging work that can be classified in this way includes messages to drive an MDB, and messages to drive a mediation. This is because MDBs and mediations always run in a servant region, in spite of the fact that the messaging engine runs in the CRA.

For further information, see:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.pmc.zseries.doc\ref\rjn0041_.html

6.6 Core group policies

The new high availability manager component of WebSphere Application Server introduced the concept of the high availability group. Every messaging engine in a service integration bus belongs to the same high availability group. The members of each group are controlled by policies that are assigned to the group at runtime.

In a cluster, you can control which servers a messaging engine can run on by configuring a policy for the messaging engine. You can configure the policy to control whether the messaging engine has a preference for a particular server, or set of servers, and whether that preference is enforced.

Static

The messaging engine can run only on a particular server, and cannot failover to any other server in the cluster. If you have multiple messaging engines this can be a useful configuration for workload balancing where failover is not wanted.

One of N

The messaging engine can failover to any of the other servers in the cluster, and has no preference for any particular server. There are two options that can modify a *One of N* policy, as follows:

- ▶ Preferred servers

The preferred servers option restricts the messaging engine to run only on the servers in the preferred servers list. The messaging engine will prefer to run on a server that is higher on the list than one that is lower down on the list.

If the *Preferred servers only* option is selected then the messaging engines can only run on servers in the list. Care must be taken with the *Preferred servers only* option as it may actually reduce availability because if none of the servers on the list are available then the messaging engine will not run. This is in spite of the fact that there may be other servers available.

- ▶ Failback

This option works with the preferred servers list. It will cause the messaging engine to automatically move to a server that it has a stronger preference for, as soon as it becomes available.

No operation

The messaging engine is managed by an external high availability framework and can failover to any of the other servers in the external high availability cluster.

Match criteria

Using the match criteria you have the ability to associate the policy with all messaging engines, all messaging engines on a named bus, all messaging engines in a particular cluster, or a single messaging engine with a specific name. The details are contained in.

Table 6-2 Match criteria

Name	Value	The messaging engines that the policy will match
type	WSAF_SIB	Any messaging engine
WSAF_SIB_MESSAGING_ENGINE	The name of the messaging engine	A particular messaging engine
WSAF_SIB_BUS	The name of the bus	All messaging engines in a particular bus
iBM_ham_cluster	The name of the cluster	All messaging engines in a particular cluster

6.7 Things to consider when writing applications

There are a number of considerations that should be taken into account when writing a messaging application.

- ▶ Can the application tolerate lost messages?

If the application can tolerate lost messages, then consider selecting a quality of service such that when a messaging engine fails, any messages in that partition are deleted.

- ▶ If the application can tolerate lost messages, does the application depend on processing messages in the order that they arrive at the destination?

If the application depends on processing messages in the order they arrive, then messaging engine clustering is unsuitable. Consecutive messages can be delivered to different messaging engines, consequently the order in which they are processed is unpredictable.

- ▶ If the application does not depend on processing messages in the order that they arrive, can the application tolerate some messages being delayed while a messaging engine restarts?

If the application can tolerate this type of delay, then messages being temporarily unavailable due to failover won't be a problem.



Part 2

System setup

In this part we describe how we set up the high-availability infrastructure. First we show the Linux for zSeries system setup, then we discuss the back-end z/OS infrastructure setup.

Archived



Setting up the infrastructure

In this chapter, we provide a detailed description of the infrastructure we implemented during our project.

7.1 Infrastructure considerations

For our architecture, our goal was high availability and scalability. The object of high availability is to have no single point of failure—therefore, in our back-end setup we decided to separate various components onto separate LPARs spread across two physical machines. It is good practice to place these LPARs on *separate* machines to ensure that an individual piece of hardware is not a single point of failure.

In our front-end setup, due to some limitations in available hardware, we decided to place our Linux machines on different LPARs on a *single* machine. We do recognize, however, that real live environments may call for a highly available front end—therefore, we will briefly describe some possible scenarios for the front-end IBM WebSphere Edge Components.

The object of scalability is to have a reasonable way to expand the number of load-balanced servers, and so we decided to use certain technologies that would allow us to do that. For example, in the case of the front-end architecture, we utilized z/VM and Linux as a virtualization technology (Linux for zSeries) to create servers as needed.

Security was not a focus for our setup. In a real environment, you will need to consider various aspects of security, including securing connections over HTTP as well as a DMZ (De-Militarized Zone).

In earlier redbooks, we discussed setting up a DMZ in more detail. You can start by looking at the last edition of this book, *Architecting High Availability e-business on IBM eServer zSeries*, SG24-6850.

Note: The previous level of this redbook is available as a part of the additional materials for this redbook. See the redbook site for details.

In the last book, as an optional exercise, we added a proxy cluster to our configuration to demonstrate how you could use a proxy to cache certain requests. In this book, since the focus is on the back-end high availability, we skip the proxy discussions.

7.2 The architecture we used

Figure 7-1 illustrates the overall architecture that we used in this project.

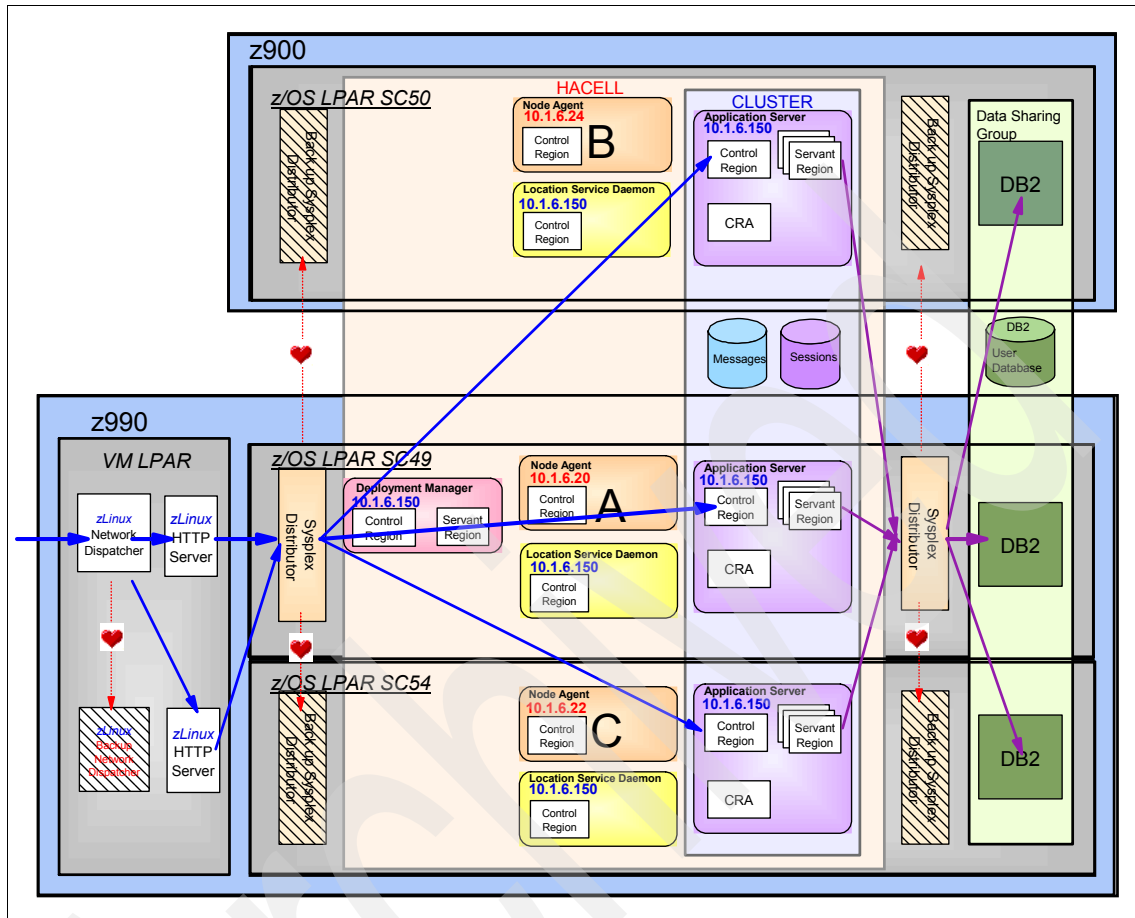


Figure 7-1 Overview of all the systems in our infrastructure

7.2.1 The front-end systems

Figure 7-2 illustrates the architecture of our front end. In the following sections, we discuss the following components in detail:

- ▶ Network Dispatcher/Load Balancer
- ▶ The IBM HTTP servers
- ▶ The WebSphere Plug-in

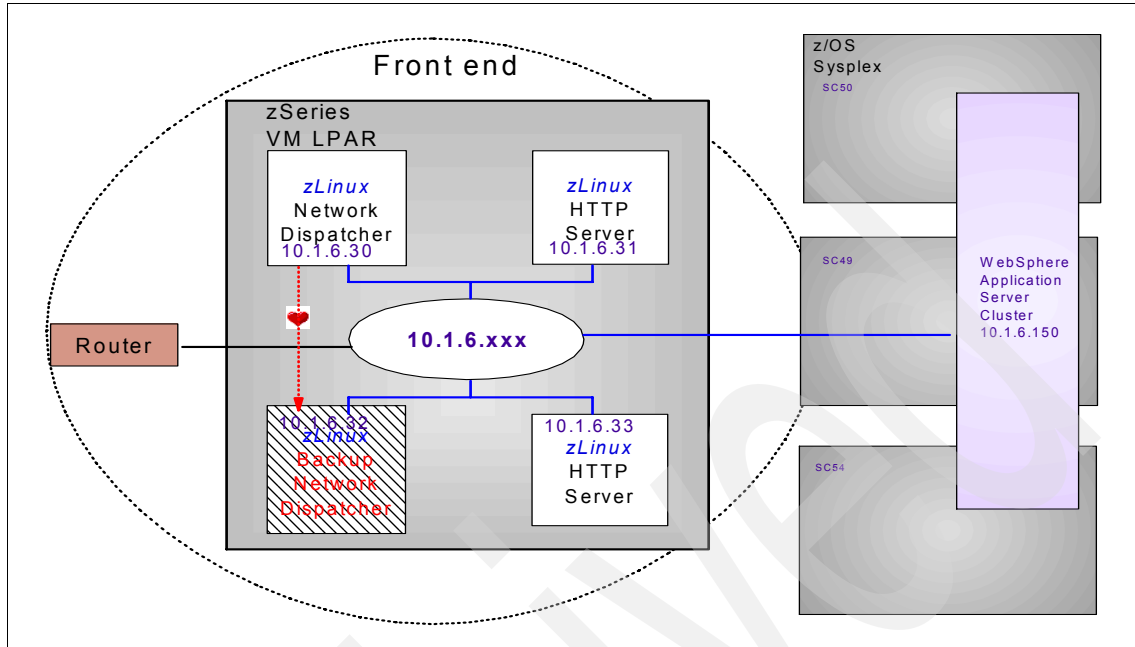


Figure 7-2 Front-end architecture Linux/zSeries

In our front-end configuration, there are a total of four Linux server machines running in the same VM (Virtual Machine) LPAR. On this LPAR, there are two Network Dispatcher machines and two IHS (IBM HTTP Server machines).

The LPAR configuration is running z/VM V5.1 connected to real OSA-E. There are a number of network configurations that can be set up, and you need to be careful in choosing a setup for which the Network Dispatchers will work properly.

We chose to connect all of our Linux machines to real OSA-E due to the limitations in hardware resources that were available to us. However, we will be looking at some of these other possible configurations, especially in the case where the Linux machines need to be on separate physical machines.

To understand some of the networking technologies that are available for the zSeries, see *Networking Overview for Linux on zSeries*, REDP3901.

We decided, for the sake of ease and simplicity, to host all of our machines, both the front and back end in the same subnet area. In a real live environment where security and performance is a concern, considerations should be taken to separate high-traffic components into different subnets and use proxy/cache servers in various places to serve static or pseudo static content.

Network Dispatcher heartbeat

The Network Dispatcher machines have load balancers from WebSphere Edge Components. As with the previous version of this book, the backup dispatcher machine will be monitoring the up-time status of the primary dispatcher machine through what is commonly known as a heartbeat. If the back dispatcher machine detects that the first machine is down, it issues the appropriate command to take over the cluster IP address of the primary server.

Note: IP takeover does not happen in the same way as it does in “plain old Ethernet” configuration, where a gratuitous arp is issued to the router telling it where to send its packets. Since our OSA-E configuration only allows for Layer 3 transport at this time (not Layer 2, which is traditional), we only had the option of using the **goActive/goStandby** scripts that we will look at a bit later.

Isolating heartbeat

As shown in Figure 7-2 on page 152, we decided to bring up a CTC connection on each dispatcher machine. We did this to isolate heartbeat traffic away from the load balancing traffic. It’s generally a good idea to have your heartbeat separate from the main subnet body.

The IHS machines have HTTP servers, which were also obtained from WebSphere Edge Components. We set up two IHS machines, along with the WebSphere V6 plug-in, and allowed the Network Dispatcher to load-balance traffic to them.

7.2.2 The back-end systems

On the back-end systems, we installed the WebSphere Application Server for z/OS V6 on three system LPAR images (SC49, SC50, and SC54). The plug-in that had been installed on each of the IHS servers will communicate to the back end via a VIPA cluster address through a Sysplex Distributor.

We describe the implementation of each component in the following sections.

As shown in Figure 7-3 on page 154, we installed a clustered application server, spread across three LPARs on two physical machines. We also installed DB2 in data-sharing mode across the same three LPARs.

The reason for having at least three z/OS systems is to maintain high availability; if one system fails or is taken down for maintenance reasons, a Parallel Sysplex is still available, with two active systems.

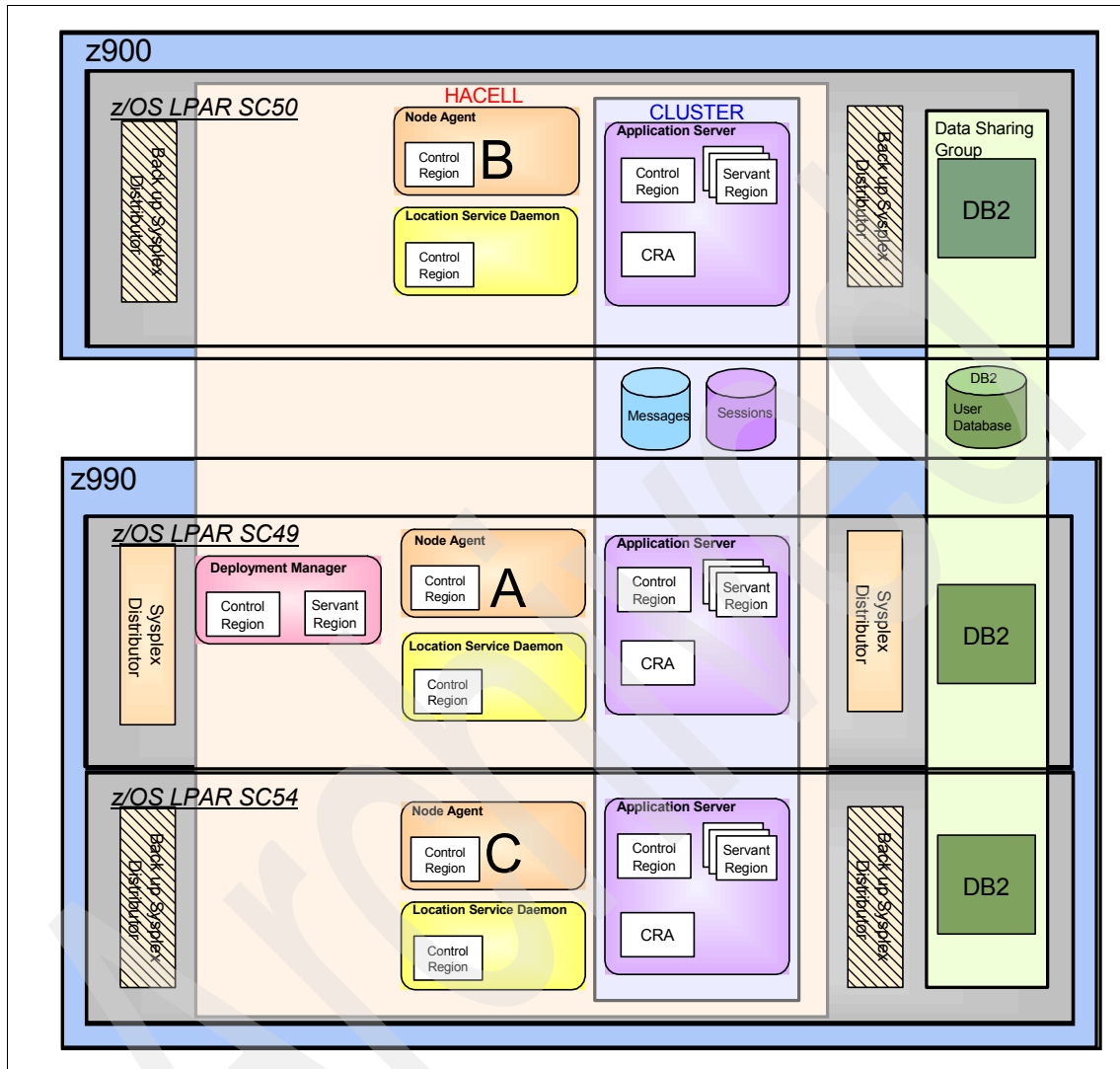


Figure 7-3 Overview of back-end systems

Production system

The failover capability will be retained, both if maintenance is performed on one system, and if an unplanned outage occurs of some component on the Parallel Sysplex. This environment represents our production environment.

Maintenance and test systems

In this redbook, we do not refer to the maintenance system and the test or integration systems. In a real customer environment, however, there should be a test sysplex consisting of at least two systems and a maintenance system.

It is reasonable to have a Parallel Sysplex for testing and not just a single system, in order to be as close as possible to the production environment.

The following hardware and software components are relevant for the high availability of the WebSphere Application server for z/OS.

Table 7-1 lists the release levels of the products we used in our environment.

- ▶ Coupling Facility
We had two external Coupling Facilities (CF) installed.
- ▶ Open System Adapter
For networking connections, we used three Open System Adapter (OSA) adapters.
- ▶ WebSphere Application Server V 6.0.1 for z/OS
- ▶ TCP/IP with Sysplex Distributor
- ▶ Resource Access Control Facility
Resource Access Control Facility (RACF) uses a sysplex-wide shared database.
- ▶ Resource Recovery System
Resource Recovery System (RRS) was used for two-phase commit.
- ▶ Workload Manager
Workload Manager (WLM) was set up in goal mode.

Table 7-1 Product levels used in this project

Product name	Release level
z/OS	V1.4
WebSphere	V6.0.1
DB2	V7.1

7.3 Setting up the Network Dispatcher

In this section we provide an overview of the general installation process for Network Dispatcher on Linux for zSeries, and describe which kernel modules we used.

7.3.1 Linux for zSeries installation

At the time of writing, we wanted to use the latest available fix pack levels and operating systems. As with any software installation, before starting it is good practice to check your documentation and release notes to see if there are any workarounds or problems that need to be addressed up front.

We built all of our Linux for zSeries servers with the following components:

- ▶ SLES 8
- ▶ SP 3 (kernel build 2.4.21-83-default)
- ▶ Timer patch (enabled)
- ▶ 512 MB of virtual storage space
- ▶ 60 MB of free space for Network Dispatcher (other components may require additional space)

To install Network Dispatcher, you can either follow a command line approach or use a GUI interface. Refer to the manual for details of each method.

Note: If you decide to use a GUI, you will need to set up X11 forwarding to render graphics on a remote display.

Linux for zSeries: kernel modules update

As we learned from the WebSphere Edge Components InfoCenter, fix pack 2 for WebSphere Edge Components V5 was needed to bring the product up to a level that would be supported on an SLES 8 (Linux/zSeries) platform.

In the previous version, we used V5.0.2 because that was available to us at that time. In this updated redbook, we used V6.0.1 because we wanted to be at the latest level with fixes.

Note: For our purposes, nothing much has changed in terms of new features between V5 and V6. However, we decided to go with the latest upgrade and patch level that was available to us. You can read more about what is available in the V6 release in *Load Balancer Administration Guide Version 6.0.1*, GC31-6858, p 6-7.

If the Dispatcher distributor fails to load, and you receive the message “unable to load kernel”, then you will need to either contact IBM support and ask them to build a kernel module that will match yours, or recompile (or load) a kernel to match the support levels that come with your WebSphere Edge Components software package. You can determine which kernel version you are running by issuing the command `uname -r` on your Linux machine.

By listing the kernel modules in the bin directory for the load balancer, you can see which kernel modules you have; see Figure 7-4 for a sample listing. Again, if you do not want to recompile (or reload) another kernel version, you will need to contact IBM WebSphere Edge support to ask them to build you a kernel module that will match your kernel version.

```
lnxes1:/opt/ibm/edge/lb/servers/bin # uname -r
2.4.21-83-default
lnxes1:/opt/ibm/edge/lb/servers/bin # ls
..                               ibmlb-2.4.21-20.0.1.EL-s390  ibmlb-2.4.21-83-default
goActive                         ibmlb-2.4.21-20.EL-s390     ibmlb-2.4.21-9.0.3.EL-s390
goInOp                           ibmlb-2.4.21-216-default   ibmlb-2.4.21-9.EL-s390
goStandby                        ibmlb-2.4.21-241-default   ibmlb-2.6.5-7.111-s390.ko
ibmlb-2.4.19-3suse-SMP           ibmlb-2.4.21-251-default   ibmlb-2.6.5-7.135-s390.ko
ibmlb-2.4.19-3suse-SMP-70       ibmlb-2.4.21-266-default   ibmlb-2.6.5-7.145-s390.ko
ibmlb-2.4.19-4suse-SMP         ibmlb-2.4.21-27.0.1.EL-s390  ibmlb-2.6.9-5.EL.ko
ibmlb-2.4.21-15.0.2.EL-s390     ibmlb-2.4.21-27.0.2.EL-s390  lbpd
ibmlb-2.4.21-15.0.4.EL-s390     ibmlb-2.4.21-27.EL-s390     loaded_version
ibmlb-2.4.21-15.EL-s390        ibmlb-2.4.21-273-default    lxexecutor
ibmlb-2.4.21-4.EL-s390
lnxes1:/opt/ibm/edge/lb/servers/bin # █
```

Figure 7-4 Kernel modules listing

In our case, since we already had the 2.4.21-83-default module, we were okay to continue with V6.0.1.

Configuring network adapter interfaces/cluster IPs

In our SLES 8 environment, we shared an OSA card among each of our four Linux servers, as depicted in Figure 7-2 on page 152. Specific instructions to configure the definition details of your network interfaces should be found in your SLES 8 documentation.

A heartbeat configuration was set up between the two dispatcher machines to communicate with each other on port 12496 via a CTC point-to-point connection. We chose port 12496 because no other service on our Linux dispatcher machines was using this port at the time.

We also needed to configure a cluster IP. A *cluster* is a collection of servers that are all accessible over one address, the cluster address. As you will see, this cluster address is important for High Availability (HA) failover support. The cluster address was 10.1.6.34.

The cluster address is not an interface address—it is an *alias* to an interface. The clients use this address as their destination address.

An alias to an interface means an additional IP “piggy-backing” on the main interface. Initially, on the primary Network Dispatcher machine, we configured our cluster IP address as an alias to the network interface listening on 10.1.6.34. To do this, we used the following command:

```
ifconfig eth1:0 10.1.6.34 netmask 255.255.255.255 up
```

The :0 in eth1:0 tells ETH1 that <cluster_ip> should be added as an alias. Similarly, on the backup Network Dispatcher machine and *all* IHS servers, we configured our cluster IP address as an alias to the loopback interface. To do this, we used the following command:

```
ifconfig lo:0 10.1.6.34 netmask 255.255.255.255 up
```

Note: Configure both Dispatcher machines to load the alias to the loopback interface on reboot. To do this, create a file named `ifcfg-lo:1` and place it in the `/etc/sysconfig/network/` directory.

We did this because our tests seemed to indicate that failover performs better when the loopback alias is already loaded before the Network Dispatcher runs the `goActive` script. The reason for this is that, when the primary dispatcher goes down, “lost” packets are captured by the backup dispatcher system and immediately processed in the TCP stack.

Configuring the loopback on the IHS servers is necessary for the same reason— the forwarding cluster IP address isn’t altered when the dispatcher forwards packets to IHS. IHS is then able to accept the packet and process it since the destination IP address is carried on the loopback. This aliasing on the loopback is part of the reason why MAC forwarding is possible on Linux.

Example 7-1 shows what the `ifconfig` output looks like on the primary dispatcher machine (LNxes1).

Example 7-1 Output of ifconfig on primary dispatcher machine

```
lnxes1:/ # ifconfig
ctc0      Link encap:Serial Line IP
          inet addr:192.1.6.38 P-t-P:192.1.6.38 Mask:255.255.254.0
          UP POINTOPOINT RUNNING NOARP MTU:32760 Metric:1
```

```

RX packets:873854 errors:0 dropped:0 overruns:0 frame:0
TX packets:874178 errors:1 dropped:1 overruns:0 carrier:1
collisions:0 txqueuelen:100
RX bytes:34955172 (33.3 Mb)  TX bytes:38492844 (36.7 Mb)

eth1    Link encap:Ethernet  HWaddr 00:09:6B:1A:1F:19
        inet addr:10.1.6.30  Bcast:10.1.6.255  Mask:255.255.255.0
        inet6 addr: fe80::9:6b00:111a:1f19/64  Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1492  Metric:1
        RX packets:55395 errors:0 dropped:0 overruns:0 frame:0
        TX packets:55466 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:3152229 (3.0 Mb)  TX bytes:4506910 (4.2 Mb)

eth1:0  Link encap:Ethernet  HWaddr 00:09:6B:1A:1F:19
        inet addr:10.1.6.34  Bcast:10.255.255.255
Mask:255.255.255.255
        UP BROADCAST RUNNING MULTICAST  MTU:1492  Metric:1

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128  Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:374 errors:0 dropped:0 overruns:0 frame:0
        TX packets:374 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:36767 (35.9 Kb)  TX bytes:36767 (35.9 Kb)

```

Configuring Network Dispatcher to load on reboot

We configured both our primary and secondary dispatcher machines to load the Network Dispatcher server (via the command `dsserver`) on reboot. There are many ways you can do this, but we suggest configuring the run-level directories to run a script on reboot. For more information, you can refer to the *init.d* man pages for Linux by issuing the `man init.d` command.

7.3.2 Configuration methods

You can define your basic configuration by using any of these methods:

- ▶ Graphical user interface (GUI)
- ▶ Configuration wizard
- ▶ Command line
- ▶ Scripts

For this project, we used scripts because this method made it easier to reload changes on the fly. You may use the GUI, but for a Linux for zSeries system, since you generally do not have a graphical display, you will need to consider how you will render these graphics on a remote machine. For more information on using these methods, refer to *Load Balancer Administration Guide Version 6.0.1*, GC31-6858.

After you finish the configuration, copy the changes to the default.cfg file so that the Dispatcher can load these changes on reboot. Example 7-2 shows the script that we used for the primary Dispatcher.

Example 7-2 default.cfg (on the primary Dispatcher machine)

```
dscontrol executor start

#Load Balancer configuration
dscontrol executor set nfa 192.1.6.38
dscontrol cluster add 10.1.6.34
dscontrol port add 10.1.6.34:80
dscontrol server add 10.1.6.34:80:10.1.6.31
dscontrol server add 10.1.6.34:80:10.1.6.33
dscontrol manager start manager
dscontrol advisor start Http 80

#High Availability configuration
dscontrol highavailability heartbeat add 192.1.6.38 192.1.6.39
dscontrol highavailability backup add primary auto 12426
```

Example 7-3 shows the script that we used for the backup Dispatcher.

Example 7-3 default.cfg (on the backup Dispatcher machine)

```
dscontrol executor start

#Load Balancer configuration
dscontrol executor set nfa 10.1.6.39
dscontrol cluster add 10.1.6.34
dscontrol port add 10.1.6.34:80
dscontrol server add 10.1.6.34:80:10.1.6.31
dscontrol server add 10.1.6.34:80:10.1.6.33
dscontrol manager start manager
dscontrol advisor start Http 80

#High Availability configuration
dscontrol highavailability heartbeat add 10.1.6.39 10.1.6.38
```


7.3.3 Load Balancer configuration

In this section, we provide a general explanation of the Load Balancer and high availability; for more detailed information, refer to *Load Balancer Administration Guide Version 6.0.1*, GC31-6858.

From the configuration script (Example 7-3), you see that we added a cluster IP of 10.1.6.34 and configured the Network Dispatcher's executor to use it. We also gave specific information to the executor to allow traffic to be forwarded through port 80 to our IHS servers.

We also implemented an HTTP advisor. This function monitors each server on the defined port. Based on the advisor information, the requests are distributed to each Web server.

The advisor will recognize if one of the Web servers fails. When this happens, the Dispatcher machine will not send requests to this server until the server comes back again. The HTTP advisor is one of many functions the Load Balancer uses to determine where to send the traffic.

7.3.4 High availability configuration

Network Dispatcher provides a built-in high availability function. The high availability environment involves two Network Dispatcher machines with connectivity to the same clients and to the same cluster of servers, as well as connectivity between themselves. Both Network Dispatcher servers must be using the same operating systems, and they must be connected to the same network.

A Network Dispatcher cluster and its cluster address can be active on just one server at a time. So the high availability configuration consists of a primary and a backup (standby) server.

The backup machine, configured in a very similar way to the primary machine, stays in standby mode unless the primary machine fails. The two machines are synchronized (which means the active server sends the connection table to the standby server). Only the primary machine routes packets, while the backup machine is continually being updated.

The two machines establish communication to monitor each other's status (referred to as a *heartbeat*), using a port that you can choose.

The standby (backup) machine uses the heartbeat and multiple reachability criteria to decide if a failure has occurred on the active (primary) server. If the primary machine fails, the backup machine detects this failure, switches to active state, and begins taking over the routing of packets.

We implemented only the heartbeat monitor in our configuration and connected the machines via a channel-to-channel point-to-point link. Both WebSphere Edge servers and both Web servers were in the same OSA-E segment.

Note: If your underlying TCP/IP VM configuration is Layer 3, the WebSphere Edge server and load balancers must be sharing the same OSA-E card for the MAC forwarding method to work.

Recovery Strategy

Next we had to define a recovery strategy. Two strategies are available, auto recovery and manual recovery:

- ▶ Auto recovery

With auto recovery, if the primary server fails and then becomes operational again, it will automatically become the active server again.

- ▶ Manual recovery

With manual recovery, the primary server remains in standby after becoming operational, and you have to manually change its status to active.

When the primary machine is operational again, but in standby state, you can either decide that it automatically becomes the active machine, or leave it in standby mode. (If you choose the second case, you will have to manually change its status if you later want it to become the active machine again.)

We decided to use auto recovery, so that each time the failed primary server (LNXES1) comes back, it becomes the active Network Dispatcher server in our configuration. Example 7-2 and Example 7-3 on page 160 illustrate this configuration for high availability (refer to the `dscontrol highavailability` tags).

If the primary server fails, the cluster address must be taken over by the backup server. The cluster address is an alias to the network interface of the active server. In the standby machine, the cluster address must be aliased to a loopback device.

Since the Dispatcher machines switch states when a failure occurs, the alias configuration must be changed automatically. In order to achieve this, Network Dispatcher executes user-created scripts.

Sample scripts can be found in the ...lb/servers/samples directory and must be copied to the ...lb/servers/bin directory in order to run. You have to modify only your cluster address, netmask, and network interface to the sample scripts.

For our high availability scenario, four scripts are important:

- goActive** Executes when a Dispatcher goes into active state and begins routing packets.
- goStandby** Executes when a Dispatcher goes into standby state monitoring the health of the active machine, but not routing any packets.
- goInOp** Executes when a Dispatcher executor is stopped and before it is started for the first time.
- highavailChange** Executes on both the primary and backup Dispatcher machines to send an e-mail notification to an administrator that a failover took place.

Example 7-4 shows our goActive script.

Example 7-4 goActive script

```
CLUSTER=10.1.6.34
INTERFACE=eth1:0
NETMASK=255.255.255.255

echo "Deleting the loopback alias"
ifconfig lo:0 down

echo "Adding the device alias"
ifconfig $INTERFACE $CLUSTER netmask $NETMASK up
```

Example 7-5 shows our goStandby script.

Example 7-5 goStandby script

```
CLUSTER=10.1.6.34
INTERFACE=eth1:0
NETMASK=255.255.255.255

echo "Deleting the device alias"
ifconfig $INTERFACE down

echo "Adding the loopback alias"
ifconfig lo:0 $INTERFACE netmask $NETMASK up
```

Example 7-6 shows our go InOp script.

Example 7-6 goInOp script

```
INTERFACE=eth1:0

echo "Deleting the loopback alias"
ifconfig lo:0 down

echo "Deleting the device alias"
ifconfig $INTERFACE down
```

After implementing the basic and high availability configurations, we intensively tested the takeover function. It worked as expected; the standby server took over the workload balancing.

7.3.5 Web server forwarding configuration

MAC forwarding is turned on automatically with the scripts we used in Example 7-2 and Example 7-3 on page 160. When using the MAC forwarding method, the Network Dispatcher will only work with back-end servers that allow the loopback adapter to be configured with an additional IP address, for which the back-end server will never respond to address resolution protocol (ARP) requests.

Using the MAC forwarding method, the Dispatcher component does not change the destination IP address in the TCP/IP packet before forwarding the packet to a TCP server machine (in our case, the Web server). By setting or aliasing the loopback device to the cluster address, the load balanced server machines will accept a packet that was addressed to the cluster address.

If you have an operating system that supports network interface aliasing (such as AIX, Linux, Solaris™, or Windows® 2000), you should alias the loopback device to the cluster address. The benefit of using an operating system that supports aliases is that you have the ability to configure the load balanced server machines to serve multiple cluster addresses.

Note: The documentation for WebSphere's load balancer indicates that you may need to apply certain patches to your Linux kernel to suppress ARP responses on the loopback device. It turns out that for a Linux for zSeries system configured with an underlying Layer 3 network, you do not need to do this in some configurations (such as the one used in our project); refer to "7.3.6, "z/Series TCP configuration (Layer 2 vrs Layer 3)" on page 165" for further details.

Our edge servers are Linux for zSeries systems. So we defined our cluster address (10.1.6.34) as an alias to the loopback device. Since our Linux systems already had loopback devices installed (this comes with a default installation of SLES), we did not need to define any.

7.3.6 z/Series TCP configuration (Layer 2 vrs Layer 3)

In *Architecting High Availability e-business on IBM eServer zSeries*, SG24-6850, we used the TCP/IP configuration Layer 3 ability. Since that time, Layer 2 has become available as another option for us to use. However, for our purposes, we decided to continue to use Layer 3. The reason for this was twofold:

- ▶ Hardware limitations

For complete high availability, we should have separated our network dispatcher components on to two z990s, using VSWITCH Layer 2. Since we didn't have two z990s, we put both network dispatcher machines on one machine.

- ▶ Performance

While Layer 3 must process frames and ARP on the OSA card, Layer 2 also must spend CPU cycles to look up or add or remove the MAC header at the host. It seems that the amount of processing is the same.

We investigated some performance testing results; the following link will provide more information:

<http://www.vm.ibm.com/perf/reports/zvm/html/layer2.html>

It was generally observed that as the traffic load increases, Layer 2 may see a bit better performance than Layer 3, but not by much.

Therefore, since we would not be running a significant load for our testing purposes and due to limitations in hardware, we opted for Layer 3.

Note: If you wish to take a look at the ARP caches on the OSA, you can issue the `qetharp -nq <interface>` command.

TCP/IP configuration issues on s390 and zSeries platforms

Configuring your load balancer to work on the s390 and zSeries machines can be complicated since the number of TCP networking permutations is quite high. Unlike traditional “plain old Ethernet” you need to consider and understand various scenarios, especially when planning for high availability with Linux.

You will often need to try a number of configurations with TCP before you get to a point where you feel comfortable that it works best for your situation.

The following provides you with some information that we have been able to obtain and test when considering to use MAC forwarding.

When using MAC forwarding, the Load Balancer and back-end IHS components must all be on the same network segment, regardless of the platform. If any active network devices are present between these components (such as routers, bridges, tunnels, etc.) the Load Balancer will not forward traffic properly to the HTTP server, since these network devices modify the link layer headers in the TCP packet.

On a Linux for zSeries platform, the OSA (qeth driver) operates differently than most network cards. The OSA card has its own virtual “link layer” implementation, which is presented to the hosts behind it. This link layer implementation does not work in the same way as it does with “plain old Ethernet” because a number of Linux machines may share this one OSA card; hence, MAC addressing is handled differently.

Since most applications work on knowledge of just the IP layer, we just generally accept that all functions related to the link layer with MAC addressing and ARP resolution are handled by the OSA card itself.

Another function of the OSA is that it delivers IP packets to hosts based on “ifconfig’ing” the IP address to the qeth card. Effectively, the OSA card is a (bridged) network segment unto itself.

The way that Load Balancer forwards packets works across one of these “OSA bridges.” This OSA bridge has knowledge of the cluster IP we have chosen to use as well as the (virtual) MAC address schemes of the IHS hosts that are connected to the same *ethernet* segment. Therefore, the Load Balancer can MAC-forward across one OSA bridge.

This does not work when the IHS server is on a 2nd OSA card. The OSA bridge for the IHS server advertises the OSA MAC address for the server IP, but when a packet arrives from the first OSA bridge with the address of the server's OSA and the IP of the cluster, the server's OSA card does not know which of its hosts, if any, should receive that packet. The same principles that permit MAC forwarding to work across one OSA bridge do not hold when trying to forward across a second OSA bridge.

SOLUTIONS

There are a few approaches you can take to fix this problem, using platform features or using the Load Balancer (LB) features.

- ▶ If the LB and servers are on the same zSeries machine, then you can define point-to-point (ctc or iucv) connections between the LB and each server. Give the endpoints private IP addresses—the point-to-point connection will be

used only for LB-to-server traffic. Then add the servers by the IP of the server end of the tunnel. With this configuration, the cluster traffic will come in through the LB OSA card and be forwarded across the point-to-point connection where the server will respond through its own default route. The response will leave via the server's OSA card, which may or may not be the same card.

- ▶ If the LB and servers are not on the same machine, a second approach would be to configure a VSWITCH in Layer 2. This method would work best when a highly available method between two machines is desired. Using Layer 2 TCP/IP switching (which is now available in z/VM 5.1 on z990) will not alter the link layers as a Layer 3 configuration would.
- ▶ If any reworking of VM networking is not possible, then you could investigate the use of the LB's Generic Routing Encapsulation (GRE) feature, a protocol that permits LB to forward across routers. When using GRE, the packet is received by LB, encapsulated, and sent to the server. At the server, the original IP packet is encapsulated, and the server responds directly to the client. GRE maintains the advantage that LB only sees the client-to-server traffic, not the server-to-client traffic. It has the disadvantage that it effectively lowers the MSS of the TCP connection due to encapsulation overhead.
- ▶ Alternately, you could use NAT—although we suggest not to do that unless your situation rules out any other alternative.

7.4 Setting up the IBM HTTP Server on zLinux

We installed two Linux for zSeries servers running the IBM HTTP Server (IHS) V6.0. This HTTP server is based on the Apache HTTP server and comes packaged with WebSphere Edge Components V6.0.1. On each IHS, we installed the plug-in which also came with WebSphere Edge Components. The Web server (or HTTP server) plug-in is the component that enables communication between the HTTP server and the WebSphere application server.

The plug-in uses the industry-standard HTTP transport protocol for non-secure transports and HTTPS for secure transports. So this plug-in can connect directly to the HTTP Transport Handler of an application server on WebSphere Application Server for z/OS.

The main value of using the WebSphere plug-in is that it can honor session affinity to a specific application server controller region (clone) on z/OS. Without this function, you are only able to honor session affinity inside of one control region.

In the following sections, we describe how to install and set up IHS and the WebSphere plug-in to work with WebSphere Application Server for z/OS and honor session affinity.

7.4.1 IBM HTTP Server configuration

In previous versions of this book, it was mentioned that you need to create an admin userid for IBM HTTP Server and update the httpd.conf file for the DNS cluster name.

For our configuration, we used the root user for the admin and did not have a DNS cluster name, so we did not perform these tasks. No further HTTP configuration should be required.

7.4.2 WebSphere Edge Components/Web server plug-in

The plug-in that comes with WebSphere V6.0.1 continues to support a clone ID. We used the clone ID in the same way. It specifies a specific application server instance in a cloned WebSphere environment, and is unique in a WebSphere cluster.

This clone ID is appended to the session cookie sent to the client. The J2EE standard name for that session cookie is JSESSIONID. A session cookie is created by WebSphere on behalf of an application request.

The JSESSIONID cookie consists of a session ID and a clone ID. The WebSphere Edge server plug-in looks for this clone ID in order to do session tracking.

The next section describes how to install and set up the Web server plug-in.

Plug-in installation

Since the WebSphere plug-in came packaged with the WebSphere Edge Components, we installed it from the WebSphere Edge package.

To install the WebSphere plug-in, complete the following steps as outlined in the manual. The install process is straightforward: we ran the install program under a **tightvnc** session, and accepted the defaults.

You'll noticed that in your httpd.conf file, you have the two lines shown in Example 7-7 appended at the end. The lines are necessary.

Example 7-7 httpd.conf changes

```
LoadModule was_ap20_module /opt/IBM/WebSphere/Plugins/bin/mod_was_ap20_http.so
WebSpherePluginConfig /opt/IBM/WebSphere/Plugins/config/webserver1/plugin-cfg.xml
```

You make changes to your plug-in configuration by updating the following file:

```
/opt/IBM/WebSphere/Plugins/config/webserver/plugin-cfg.xml
```


After you are done, you reload the changes by recycling the HTTP server:

```
/opt/IBMHTTPServer/bin/apachectl restart
```

7.4.3 Configuring the WebSphere plug-in

In order to forward requests from the WebSphere plug-in to the WebSphere Application Server for z/OS Web container (HTTP Transport Handler), you have to configure the plug-in.

For the purposes of this project, the HTTP plug-in was configured to support session affinity between the Web browser client and the z/OS WebSphere sysplex. Session affinity was achieved through the use of a clone ID. Within the plugin-cfg.xml file, we defined a <ServerCluster> tag which contained further <Server> tags defining our various WebSphere Application Servers on the sysplex.

In a configuration where the sysplex will not do any type of load balancing, defining a <ServerCluster> and various <Server> tags is all you need to do. In this type of configuration, a request that comes through to the plugin-cfg.xml will first be load balanced (by default through a round robin mechanism) to the list of server choices defined in <Server>.

Any subsequent request that comes through may or may not get load balanced across the WebSphere Application Servers defined in the <Server> tag, as explained here:

- ▶ In the case where any subsequent request carries a JSESSION cookie, it will not get load balanced, but will instead be sent directly to the IP name upon a successful match of the clone ID in the <Transport> tag.
- ▶ In the case where any subsequent request does not carry a JSESSION cookie, it will continue to be load balanced across the WebSphere Application Servers defined in the <Server> tag.

In a configuration where the sysplex is set up to do the load balancing (that is, through a Sysplex Distributor), you need to define an additional <ClusterAddress> tag to the configuration. You can assign one <ClusterAddress> tag to a <ServerCluster>, and much like a <Server> tag, it will contain the forwarding information needed.

- ▶ In this configuration, a request that comes through to the plugin-cfg.xml will first be directed to the <ClusterAddress> which consequently gets forwarded via the cluster you defined in <Transport>.
- ▶ Any subsequent request that comes from the client (and if affinity was established) will get directed to the <ServerCluster>, where the request's JSESSION cookie is matched to one of the <Server> tags to be forwarded to

the appropriate WebSphere Application Server in the sysplex. If session affinity was not established before, the request continues through the <ClusterAddress>.

In our case, we utilized the <ClusterAddress> and allowed the sysplex to handle the load balancing to its application servers. In this way, if one of the WebSphere applications goes down (because of failure or maintenance), WebSphere will assign a new JSESSIONID cookie. The plug-in will then be able to redirect subsequent responses to the backup WebSphere Application Server.

For information on how session information is kept in sync at the sysplex level and how these cookies get reassigned, refer to 4.3, “Data flow in HTTP sessions with affinity” on page 84.

The plug-in configuration file is an XML file and the location is specified in the httpd.conf file (if you have followed the instructions to this point, you will already have performed this task):

```
WebSpherePluginConfig  
/opt/IBM/WebSphere/Plugins/config/websvr/plugin-cfg.xml
```

You can use the WebSphere Administrative console to generate the plug-in, then move that file to Linux. In our case, we wanted to have a simpler plug-in for illustrative purposes, so we edited the plugin-cfg.xml file manually using a text editor.

The definitions in this file must be correlated with other definitions: the DNS, the context root, the virtual host of your application server on z/OS, and the Sysplex Distributor.

Based on our plugin-cfg.xml file, we now discuss how these definitions have to fit together; refer to Figure 7-5 on page 171.

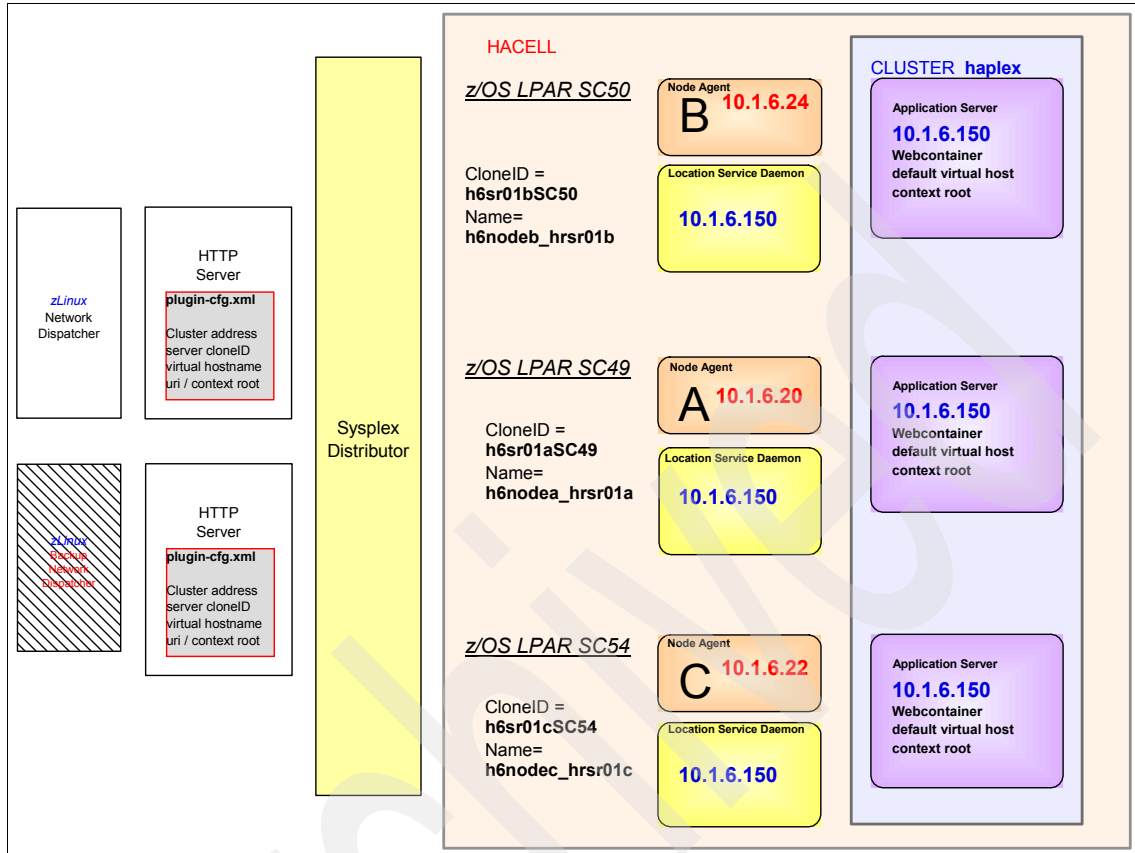


Figure 7-5 How definitions fit together

Example 7-8 Our plugin-cfg.xml file

```

<?xml version="1.0"?>
<Config>
<Log LogLevel="Error" A.
    Name="/opt/IBM/WebSphere/Plugins/config/webserver/http_plugin.1
og"/>
<ServerGroup Name="HApLex"> B.
    <ClusterAddress name="haplex"> C.
        <Transport hostname="10.1.6.150" port="39938" protocol="http"/>
    </ClusterAddress>

    <Server CloneID="h6sr01aSC49" Name="h6nodea_h6sr01a"> D.
        <Transport Hostname="10.1.6.20" Port="39938" Protocol="http"/>
    </Server>

```

```

<Server CloneID="h6sr01bSC50" Name="h6nodea_h6sr01b">D.
  <Transport Hostname="10.1.6.24" Port="39938" Protocol="http"/>
</Server>

<Server CloneID="h6sr01cSC54" Name="h6nodea_h6sr01c">D.
  <Transport Hostname="10.1.6.22" Port="39938" Protocol="http"/>
</Server>
</ServerGroup>

<VirtualHostGroup Name="default_host">E.
  <VirtualHost Name="*:80"/>F.
</VirtualHostGroup>

<UriGroup Name="default_host_URIs">G.
  <Uri Name="/TraderJMSWeb/*"/>
  <Uri Name="/TraderDBWeb/*"/>
</UriGroup>

<Route ServerGroup="HAplex" UriGroup="default_host_URIs"H.
  VirtualHostGroup="default_host"/>

</Config>

```

The following notes refer to the highlighted sections of Example 7-8.

A. **LogLevel** controls the amount of information that gets written to the plug-in log file. Possible values are: Error, Warn, and Trace. For production, Error is recommended; for debugging, use Trace.

B. **ServerGroups** provide a mechanism for grouping servers. Group all servers responsible for serving a specific application. You can define one cluster address (see **C.**) for a server group.

The connection to a specific context root and virtual host is done in the Route statement (see **H.**) We used three backend WebSphere Application Servers for our high availability test, so we grouped all instances of these servers together in a group called HAplex.

C. The **ClusterAddress** definition is a plug-in enhancement done for WebSphere Application Server for z/OS. Within the Transport definition, you can define the IP address (hostname) and port number (port) to which an HTTP request, which has no session affinity, is routed. Usually this is the distributed VIPA address of the Sysplex Distributor (SD), balancing incoming requests to your application server.

We defined IP address 10.1.6.150 and port 39938 for our server; see 8.1, “Sysplex Distributor” on page 178 for a detailed description of our SD setup.

On the `Protocol` statement, you can define whether you want to use HTTP or HTTPS for the communication between the plug-in and the Sysplex Distributor or application server.

If you choose not to use SD for workload balancing, then requests that have no session affinity are distributed round-robin to one of the servers defined in your server group.

D. On a `Server` statement, you have to define each application server instance belonging to that `ServerGroup`. A `Server` statement consists of a `CloneID`, a `Name`, and a `Transport` segment.

In WebSphere Application Server for z/OS, the clone ID is automatically generated by the Web container and has the format of `<clone_ID>`. You can find this clone ID if you open the Admin Console to WebSphere and then click **Servers** → **Application Servers** → **server name**. You will find it under the General Properties tab, in the field Unique ID.

When an HTTP request is received by the plug-in, it is scanned for a session cookie named JSESSIONID. If there is such a session cookie, it is searched for a clone ID. If a clone ID is found, it is compared to the ones defined in the server statement. If a match is found, then the request is sent to the IP address (hostname) and port defined in the `Transport` segment of the matching server. As a protocol, you can choose between HTTP and HTTPS.

With this correlation of a clone ID to a specific IP address and port, the different ports the server is listening to are hidden from the end user.

E, F. In the `VirtualHostGroup`, you have to define the hostname for which the plug-in should forward requests. It is the same kind of definition you do in the server Web container.

The `VirtualHost` Names you define in your Web container on z/OS have to be defined here, as well. All virtual hosts you want to serve on WebSphere for z/OS have to be known by the plug-in, because it has to forward these requests.

The `VirtualHost` Name is the hostname that end users use in their URL requests to connect to the application server. It can be the name which you have also defined in the Domain Name Server (DNS) resolved to the entry point of your installation, an IP address, or wildcard. In our scenarios, we used a wildcard. We used `default_host` as the `VirtualHostGroup` name.

Which application server is responsible to serve a specific hostname (virtual host) is defined in the Route statement (see **H.**)

G. At the **UriGroup Name** statement, you have to define all the context roots you want to serve in your WebSphere Application Server for z/OS. We defined the context roots of the applications we used for our tests (TraderJMSWeb and TraderDBWeb). The same context roots have to be defined in your EAR file before deploying your application.

H. A **Route** statement binds together the server (ServerGroup), the virtual host (VirtualHost Group), and the context root (UriGroup) definition.

An HTTP request contains a virtual host name and a context root name (http://10.1.6.34/TraderDBWeb/). The request is routed to the server belonging to the ServerGroup which is defined in the same Route statement as the context root and virtual host group. For this reason, you have to group the right combination in a Route statement.

In our scenario, this was easy since we had only one ServerGroup, one VirtualHost Group, and one UriGroup defined. All requests that arrived with any hostname and had one of the URIs were sent to one of the instances of WAS backend servers, depending on the clone ID in the cookie. If there was no clone ID, the request was routed to the Sysplex Distributor address and port that was defined in the `ClusterAddress` statement.

For more information about the distributed plug-in, visit the WebSphere Advanced Edition Infocenter and do a lookup on plugin-cfg.xml:

<http://www-306.ibm.com/software/webservers/appserv/ecinfocenter.html>

7.4.4 How the front-end traffic flows

In this section, we describe how the traffic is routed through the front-end architecture. Figure 7-6 shows a logical depiction of what happens when a client types http://10.1.6.34/TraderDBWeb/.

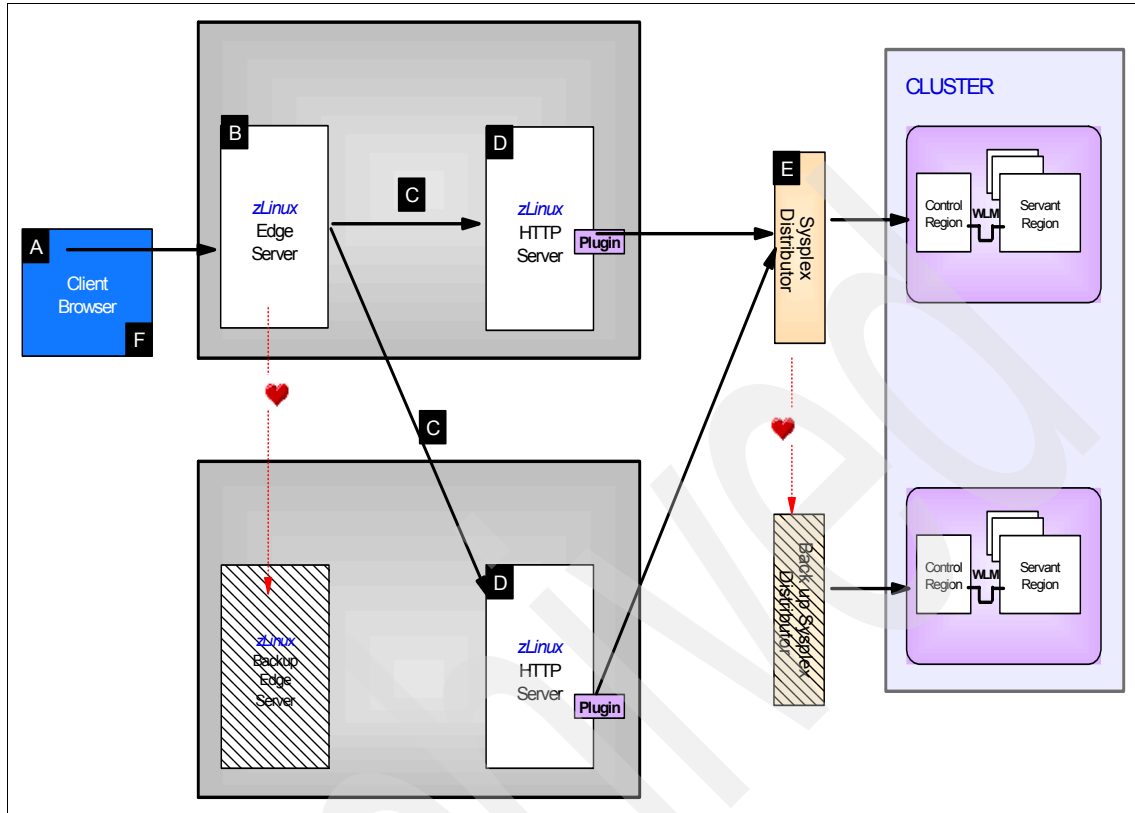


Figure 7-6 Front-end flow of traffic

Initially, at **A** the client will open a browser of their choice and enter the cluster IP address for the WebSphere Edge Server cluster located at **B**. In our scenario, and for the purposes of our testing, our browser was placed on a Windows machine residing on the 10.1.6.x ring.

The cluster IP address is currently active on the top Edge server; thus, the request will flow through the primary network dispatcher.

However, if this Edge server is down, the request will flow through the backup server, and that cluster IP address will become active on that machine (bottom system) and be routed through there.

After the packet is routed through the active WebSphere Edge server, the request will then be forwarded via **C** to one of the two IHS servers residing at **D** based upon a lookup the dispatcher makes to a weighted table.

If the first IHS server has more weight, it will win receiving the request. This weighted table is updated periodically based on the algorithm you choose to configure your load balancer with. In our scenario, we implemented a simple roundrobin solution.

Also note that there are a number of ways this request may be forwarded: the packet may be translated into another IP (NAT), the packet may be encapsulated into another datagram (GRE), or it may just be forwarded out (MAC forwarding). In our scenario, we implemented MAC forwarding.

At this point, **D**, the IHS will receive the “load-balanced” packet. The plug-in will scan this HTTP request for certain URIs. If the plug-in matches a URI, it will be forwarded to the z/OS sysplex. If the plug-in cannot match the URI, then the client will receive a 404 (file not found) response.

The Sysplex Distributor at **E** will receive the request, process it in the z/OS WebSphere sysplex, and then return it through the HTTP server via the plug-in.

Now that the packet is back to an IHS server (from which it left, depending on where it was load balanced), it will then be forwarded directly back to the client at **F**. Since MAC (or IP forwarding, in the Linux for zSeries case) had been enabled, we can do this and in our case, since our client was sitting on the 10.1.6.x ring, we just routed packets back here.

Normally—and ideally—you would want another router (separate from where the incoming traffic is coming in) sitting on the edge of your network to route traffic back out into the Internet.



Setting up the z/OS infrastructure

In this chapter, we provide a detailed description of the z/OS infrastructure that we implemented during our project.

8.1 Sysplex Distributor

We chose to implement the Sysplex Distributor because clients receive the benefits of workload distribution provided by both Workload Manager and the Quality of Service (QoS) Policy Agent. In addition, Sysplex Distributor ensures high availability of the IP applications running on the sysplex cluster, even if one physical network interface fails or an entire IP stack or z/OS LPAR is lost. Also, we can implement it on z/OS without needing additional software or hardware.

To implement Sysplex Distributor (SD), follow these steps:

1. Choose the LPAR whose IP stack will execute the Sysplex Distributor distributing function.
2. Select the LPARs containing the backup IP stacks, and in which order they should be selected.
3. Ensure that WLM GOAL mode is enabled in all the LPARs participating in the Sysplex Distributor.
4. Enable sysplex routing in all the IP stacks participating in the Sysplex Distributor with the `SYSPLEXROUTING` statement.
5. For those IP stacks that are active under a multistack environment, the same host links have to be created dynamically. In general, code `DYNAMICXCF` in all the IP stacks participating in the Sysplex Distributor.
6. Code `DATAGRAMFWD` in all IP stacks participating in the Sysplex Distributor.
7. Select, by port numbers, the applications that are going to be distributed using the Sysplex Distributor function. Note that if the application chosen requires data and control ports, both ports have to be considered.
8. Code the `VIPADYNAMIC/ENDVIPADYNAMIC` block for the distributing IP stack:
 - Define the dynamic VIPA associated to the distributing IP stack with the `VIPADEFINE` statement.
 - Associate the sysplex Dynamic VIPA to the application's port number with the `VIPADISTRIBUTE` statement.
9. Code the `VIPADYNAMIC/ENDVIPADYNAMIC` block for the distributor's backup IP stacks. Define the IP stack as backup for the sysplex DVIPA address with the `VIPABACKUP` statement.
10. VTAM® must have XCF communications enabled by specifying `XCFINIT=YES` as a startup parameter, or by activating the VTAM major node `ISTLSXCF`.

8.1.1 The implementation we used

As shown in Figure 8-1 on page 180, we implemented Sysplex Distributor (SD) on three LPARs: SC49, SC50, and SC54, with the following settings:

- ▶ System SC49 is the SD distributing stack, SC50 is backup1, and SC54 is backup2.
- ▶ The DVIPA address for our clustered application server is 10.1.6.150.
- ▶ We coded the following ports on the VIPADISTRIBUTE statement to be distributed on address 10.1.6.150:
 - 39902 and 39903, the Daemon ORB IIOp and ORB SSL ports
 - 39911 and 39912, the Deployment Manager IIOp and ORB SSL ports
 - 39938 and 39939, the w6sr01 cluster server HTTP and HTTPS ports
 - 38110, DB2 Dist ADDRESS space DRDA port
 - 7888, the DRS port
- ▶ The DVIPA has to match the hostname defined in the Clusteraddress statement of the plug-in configuration file (plug-in-cfg.xml) of the IHS server.
- ▶ We wanted the Dynamic VIPA to be taken back as soon as the original IP stack is restarted in the event of a failure. Therefore, we coded MOVEABLE IMMEDIATE.

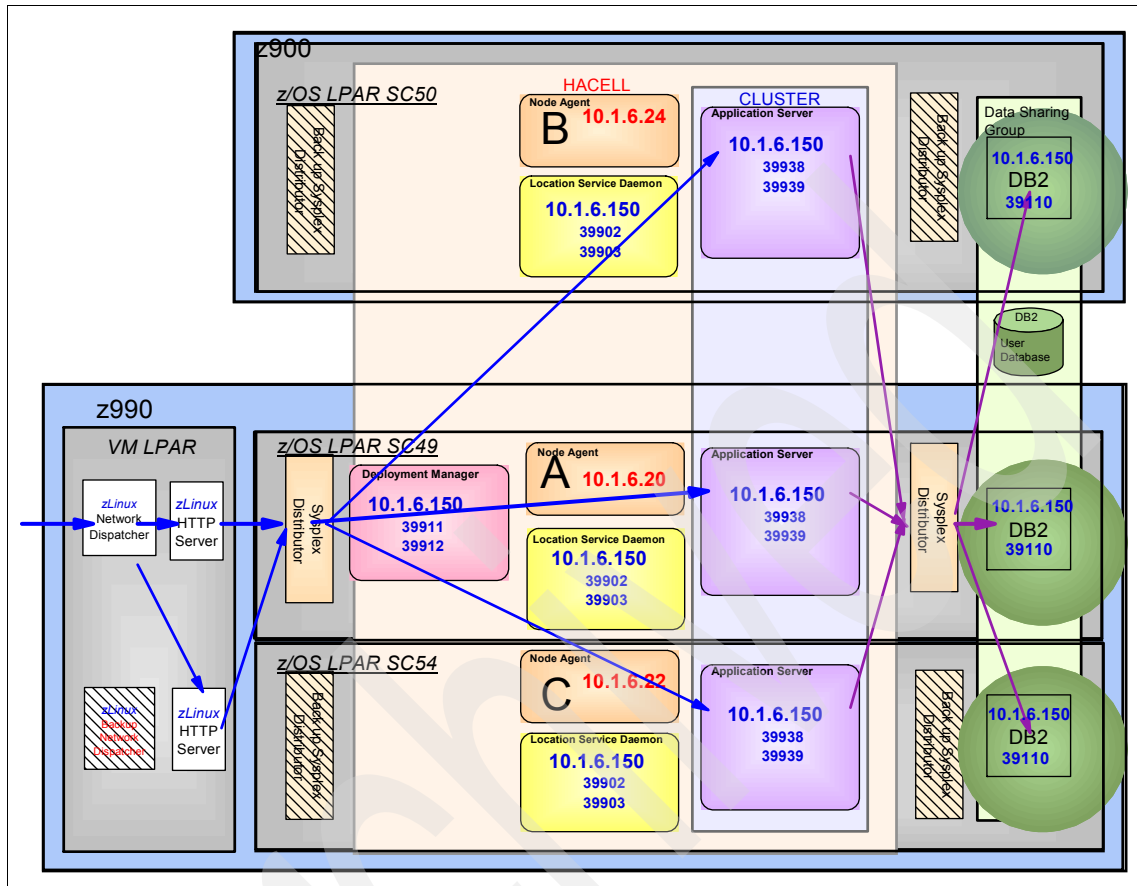


Figure 8-1 Sysplex Distributor configuration

Note: We defined the DVIPA address on the same subnet (same LAN segment) as the back-end system's IP address and the OSA,.

We performed these steps for our implementation:

1. BPXPRMxx PARMLIB changes
 - Create a temporary BPXPRMCC to add TCPIPA:
SUBFILESYSTYPE NAME(TCPIPA)
TYPE(CINET)
ENTRYPOINT(EZBPFINI)
Issue SETOMVS RESET=(CC) to add dynamically.
 - Update BPXPRMxx for the next IPL.
2. TCP/IP profile definitions
 - a. On SC49 (see Example 8-1):
 - System IP Addr= 10.1.6.20
 - Dynamic XCF address= 192.168.90.4

Example 8-1 TCP/IP profile definitions on SC49

```
;  
-----  
; Dynamic VIPA definitions:  
-----  
IPCONFIG DATAGRAMFWD VARSUBNETTING SYSPLEXROUTING SOURCEVIPA  
DYNAMICXCF 192.168.90.4 255.255.255.0 4  
;  
VIPADYNAMIC  
VIPADefine MOVE IMMEDIATE 255.255.255.0 10.1.6.150  
VIPADistribute DEFINE DISTMETHOD BASEWLM 10.1.6.150  
PORT 39911 39912 39938 39939 39902 39903  
38110 7888 80  
DESTIP 192.168.90.4 192.168.90.6 192.168.90.7  
ENDVIPADYNAMIC  
;  
-----  
; Hardware definitions:  
-----  
;  
DEVICE OSA2C60 MPCIPA  
LINK OSA2C60LNK IPAQNET OSA2C60  
;  
-----  
; HOME internet (IP) addresses of each link in the host.  
-----  
;  
HOME
```

```

10.1.6.20    OSA2C60LNK
;
;-----
; GATEWAY
;-----
;
;
BEGINROUTES
Route 10.1.6.0/24 = OSA2C60LNK MTU 1492
ENDROUTES
;
;-----
; Start all the defined devices.
;-----
;
START OSA2C60

```

b. On SC50 (see Example 8-2):

- System IP Addr= 10.1.6.24
- Dynamic XCF address= 192.168.90.6
- It is the backup1.

Example 8-2 TCP/IP profile definitions on SC50

```

;
IPCONFIG MULTIPATH IQDIOR DATAGRAMFWD VARSUBNETTING SYSPLEXROUTING
DYNAMICXCF 192.168.90.6 255.255.255.0 4
;
VIPADYNAMIC
VIPAbakup 100 10.1.6.150
ENDVIPADYNAMIC
;
DEVICE OSA2100 MPCIPA
LINK OSA2100LNK IPAQNET OSA2100
;
HOME
10.1.6.24 OSA2100LNK
;
BEGINROUTES
ROUTE 10.1.6.0/24 = OSA2100LNK MTU 1492
ENDROUTES
;
START OSA2100

```

- c. On SC54 (see Example 8-3):
 - System IP Addr= 10.1.6.22
 - Dynamic XCF address= 192.168.90.7
 - It is backup2.

Example 8-3 TCP/IP profile definitions on SC52

```

;
IPCONFIG DATAGRAMFWD VARSUBNETTING SYSPLEXROUTING
DYNAMICXCF 192.168.90.7 255.255.255.0 4
;
VIPADYNAMIC
VIPAbackup 80 10.1.6.150
ENDVIPADYNAMIC
;
DEVICE OSA2C60 MPCIPA
LINK OSA2C60LNK IPAQGNET OSA2C60
;
HOME
10.1.6.22 OSA2C60LNK
;
BEGINROUTES
ROUTE 10.1.6.0/24 = OSA2C60LNK MTU 1492
ENDROUTES
;
START OSA2C60

```

3. TCPDATA definitions

- a. On SC49 (see Example 8-4):

Example 8-4 TCPDATA definitions on SC48

```

TCPIPJOBNAME TCPIPA
HOSTNAME WTSC49A
DOMAINORIGIN ITS0.IBM.COM
LOOKUP LOCAL DNS
DATASETPREFIX TCPIPA
MESSAGECASE MIXED

```

- b. On SC50 (see Example 8-5):

Example 8-5 TCPDATA definitions on SC50

```

TCPIPJOBNAME TCPIPA

```

```
HOSTNAME WTSC50A
DOMAINORIGIN ITS0.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

c. On SC54 (see Example 8-6):

Example 8-6 TCPDATA definitions on SC54

```
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC54A
DOMAINORIGIN ITS0.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

4. Add the IP addresses and the hostnames for SC49, SC50, and SC54, and the DVIPA in /etc/hosts on the three systems (see Example 8-7):

Example 8-7 /etc/hosts

```
10.1.6.150 haplex1.itso.ibm.com haplex1
10.1.6.20 wtsc49a.itso.ibm.com wtsc49a
10.1.6.24 wtsc50a.itso.ibm.com wtsc50a
10.1.6.22 wtsc54a.itso.ibm.com wtsc54a
```

5. Check DVIPA:

Example 8-8 Check DVIP port definitions

```
D TCPIP,TCPIPA,N,VDPT
EZZ2500I NETSTAT CS V1R6 TCPIPA 155
DYNAMIC VIPA DESTINATION PORT TABLE:
DEST IPADDR      DPORT DESTXCF ADDR      RDY TOTALCONN  WLM FLG
10.1.6.150      00080 192.168.90.4    000 0000000000 02 B
10.1.6.150      00080 192.168.90.6    000 0000000000 01 B
10.1.6.150      00080 192.168.90.7    000 0000000000 02 B
10.1.6.150      07888 192.168.90.4    000 0000000000 02 B
10.1.6.150      07888 192.168.90.6    000 0000000000 01 B
10.1.6.150      07888 192.168.90.7    000 0000000000 02 B
10.1.6.150      38110 192.168.90.4    001 0000000001 02 B
10.1.6.150      38110 192.168.90.6    001 0000000000 01 B
10.1.6.150      38110 192.168.90.7    001 0000000001 02 B
10.1.6.150      39902 192.168.90.4    001 0000000000 01 B
10.1.6.150      39902 192.168.90.6    001 0000000000 01 B
10.1.6.150      39902 192.168.90.7    001 0000000000 02 B
10.1.6.150      39903 192.168.90.4    000 0000000000 01 B
```


10.1.6.150	39903	192.168.90.6	000	0000000000	01	B
10.1.6.150	39903	192.168.90.7	000	0000000000	02	B
10.1.6.150	39911	192.168.90.4	000	0000000000	02	B
10.1.6.150	39911	192.168.90.6	000	0000000000	01	B
10.1.6.150	39911	192.168.90.7	000	0000000000	02	B
10.1.6.150	39912	192.168.90.4	000	0000000000	02	B
10.1.6.150	39912	192.168.90.6	000	0000000000	01	B
10.1.6.150	39912	192.168.90.7	000	0000000000	02	B
10.1.6.150	39938	192.168.90.4	000	0000000000	02	B
10.1.6.150	39938	192.168.90.6	001	0000000000	01	B
10.1.6.150	39938	192.168.90.7	001	0000000000	02	B
10.1.6.150	39939	192.168.90.4	000	0000000000	02	B
10.1.6.150	39939	192.168.90.6	001	0000000000	01	B
10.1.6.150	39939	192.168.90.7	001	0000000000	02	B
10.1.6.150	39902	192.168.90.4	001	0000000000	01	B
10.1.6.150	39902	192.168.90.6	001	0000000000	01	B
10.1.6.150	39902	192.168.90.7	001	0000000000	02	B
10.1.6.150	39903	192.168.90.4	000	0000000000	01	B
10.1.6.150	39903	192.168.90.6	000	0000000000	01	B
10.1.6.150	39903	192.168.90.7	000	0000000000	02	B

8.2 Setting up WebSphere

In the following sections, we describe how we set up the WebSphere application server for z/OS in a Parallel Sysplex. We also briefly discuss all the other necessary software components, and the design of our HFS file system.

The objective of our configuration was to avoid a single point of failure and maximize system availability by ensuring that the configuration had some degree of redundancy. High availability topologies typically involve horizontal scaling across multiple machines.

When designing our configuration and installing WebSphere for z/OS V6, we considered the following:

- ▶ The configuration design had to support scaling to other systems, and creating additional servers. We wanted to avoid making changes like the naming convention, if the cell had to expand or do excessive additional work, if adding more servers.
- ▶ We wanted to utilize Sysplex Distributor.

8.2.1 Target infrastructure

Our environment, shown in Figure 8-2, consisted of three systems in a sysplex. We set up one cell spanning all three systems in the sysplex. The cell also contains a cluster that spans the three systems. Each node in the cell has its own Node Agent. A daemon runs for each system in the cell. The applications are deployed in the cluster.

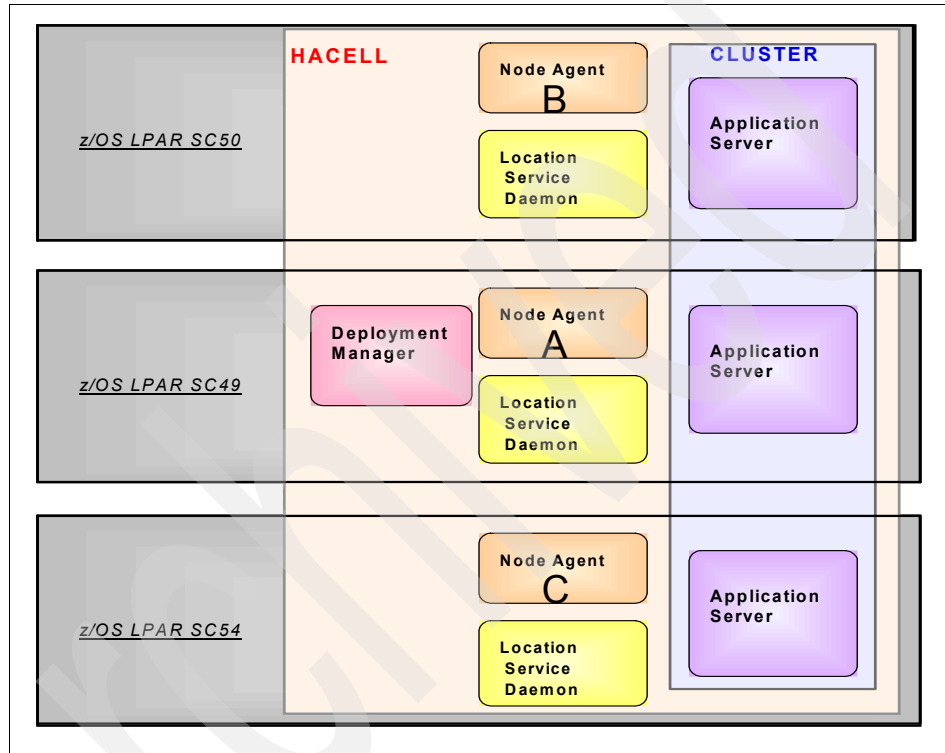


Figure 8-2 Target infrastructure

8.2.2 HFS setup for the SMP/E environment

The structure of the WebSphere code HFS files was set up in the following way to enable our maintenance strategy of a non-disruptive rolling upgrade for WebSphere for z/OS itself and for the business application servers.

We decided that each system would have its own HFS. This gave us the ability to upgrade WebSphere one at a time, and to run WebSphere with two different PTF levels at the same time.

Note: If you decide to share the HFS, you have to upgrade WebSphere on all systems at the same time.

We created three directories:

- ▶ /SC49/zWebSphere
- ▶ /SC50/zWebSphere
- ▶ /SC54/zWebSphere

These directories were mounted below the version-specific root; see Figure 8-3. We created a system-specific symbolic link named `/$SYSNAME/zWebSphere`. This link was mounted to `/usr/lpp/zWebSphere/V6R0`.

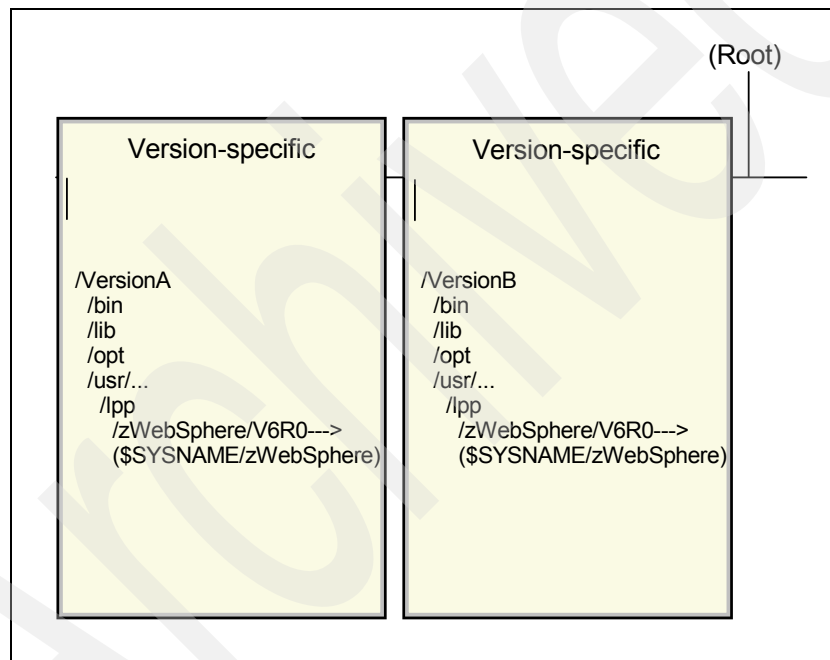


Figure 8-3 Setup of the system-specific HFS

8.2.3 Setup of the WebSphere libraries

In a Parallel Sysplex, there are two strategies concerning product libraries:

- ▶ Sharing libraries between the systems
- ▶ Keeping one set of libraries for each system

In our environment, each system had its own set of libraries. This allowed us to upgrade one system at a time.

The SMP/E maintenance system and the test environment each have a separate set of libraries.

Note: If you use a shared HFS, then you should use shared libraries, as well.

8.2.4 WebSphere configuration

Flow of installation

Figure 8-4 illustrates the flow we used for creating the H6CELL environment.

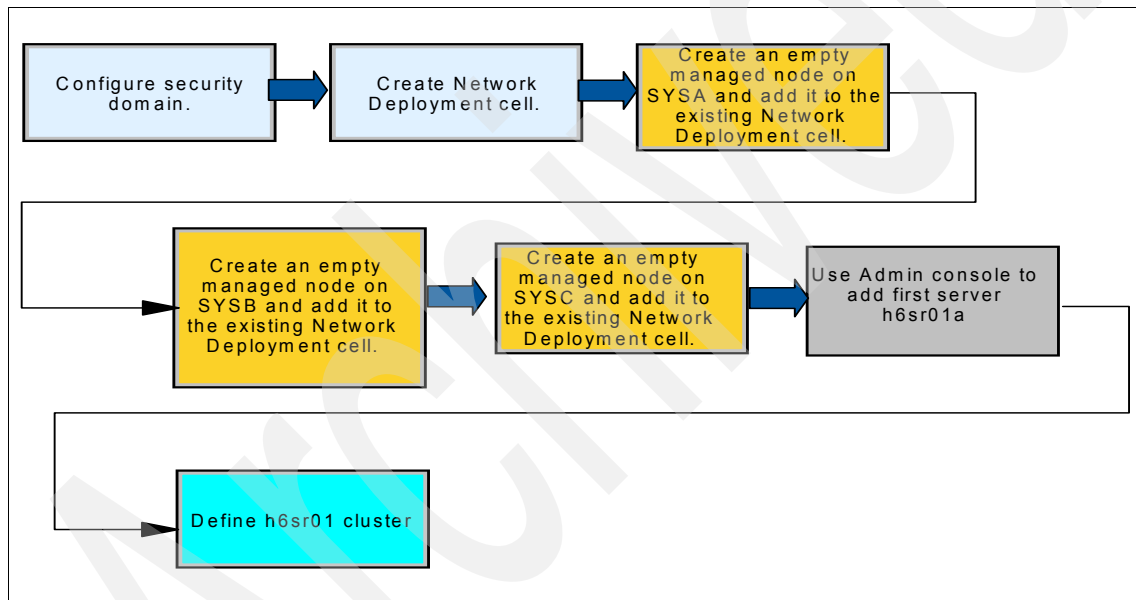


Figure 8-4 Flow of installation

The configuration HFS structure

We chose to use a separate HFS for each system in our sysplex. Each Application Server had its own configuration HFS and the Deployment Manager also used a separate HFS; see Figure 8-5 on page 189.

We created the following symbolic links:

▶ On SC49:

`/$SYSNAME/wasv6config/h6cell/h6dmnode`. This link is mounted to
`/wasv6config/h6cell/h6dmnode`

`/$SYSNAME/wasv6config/h6cell/h6node`. This link is mounted to
`/wasv6config/h6cell/h6node`

▶ On SC50 and SC54:

`/$SYSNAME/wasv6config/h6cell/h6node`. This link is mounted to
`/wasv6config/h6cell/h6node`

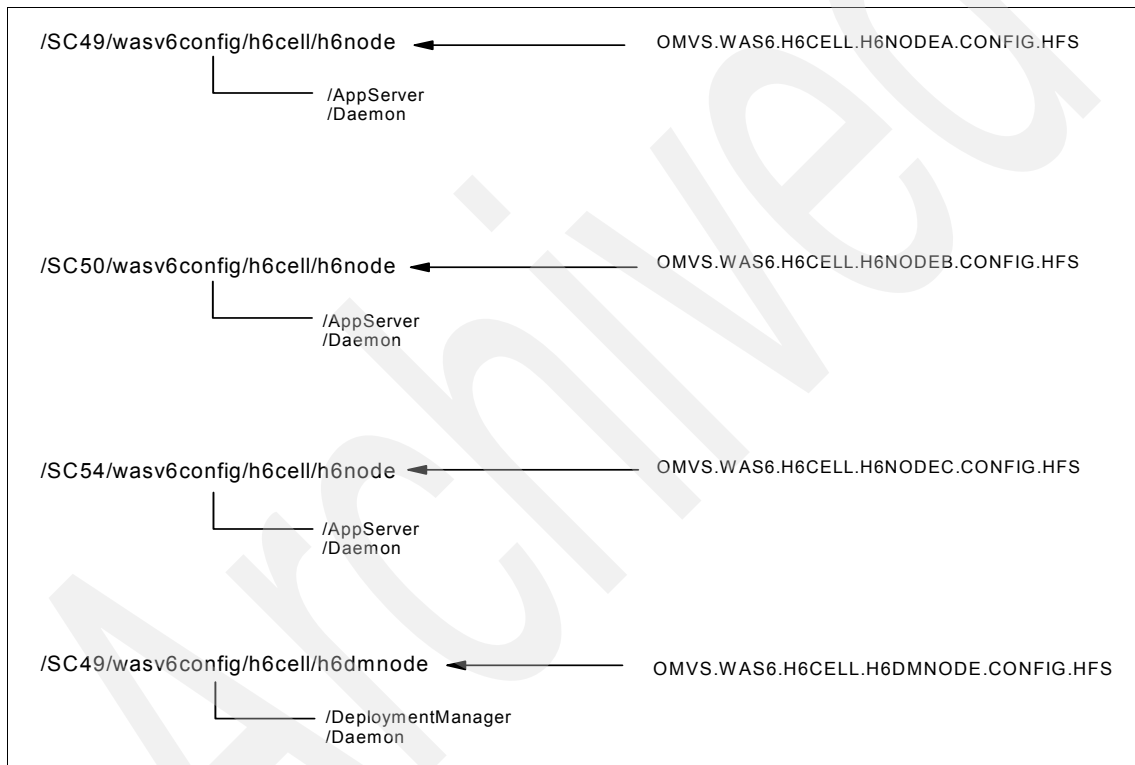


Figure 8-5 The configuration HFS structure

Naming conventions used

When we designed the naming convention, our objectives were:

- ▶ Design a naming convention that is meaningful and flexible.
- ▶ Minimize the number of JCL start procedures.
- ▶ Minimize the number of RACF userid and groups.

Figure 8-6 shows the naming convention we used.

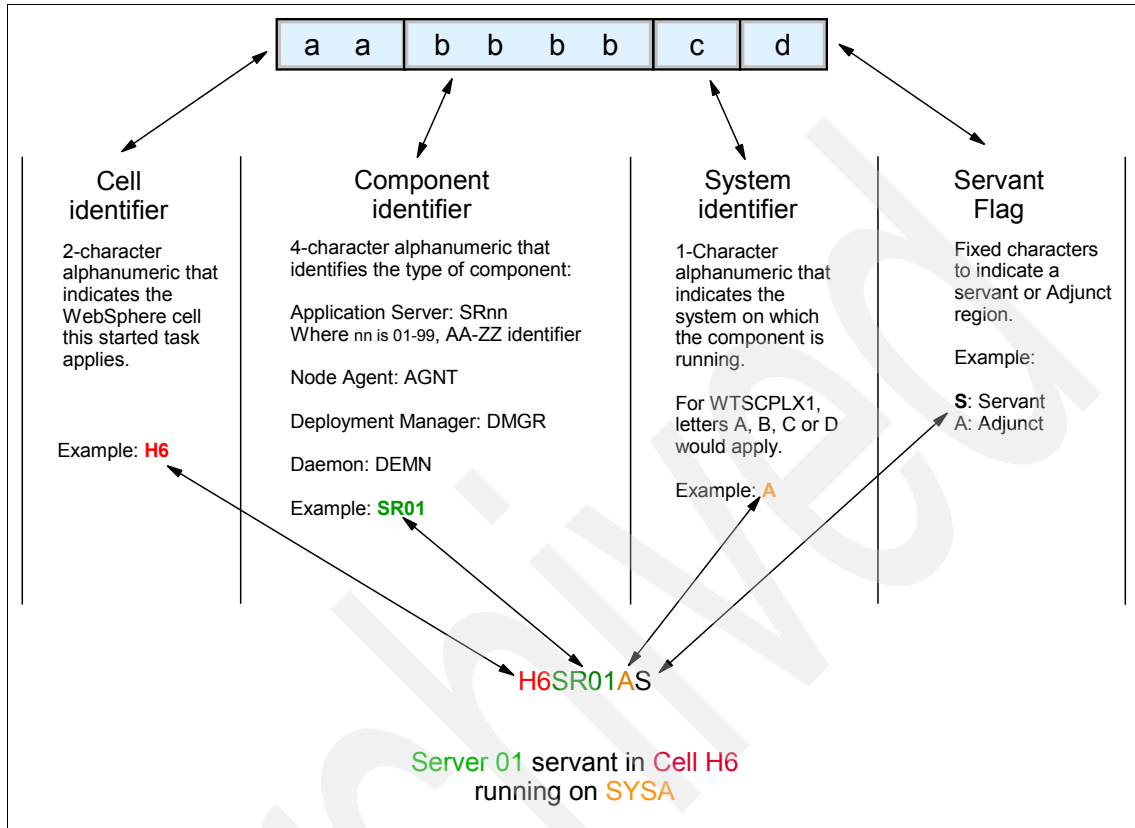


Figure 8-6 Naming convention

The following are the values we used to configure the empty managed nodes and Deployment Manager. See also 8.2.6, "Port definitions and assignments" on page 196.

Security Domain

```
WebSphere Application Server Configuration Group Information
Group....: H6CFG          UID...: 5500

WebSphere Application Server Administrator Information
User ID..: H6ADMIN      UID...: 5000
Password.: H6ADMIN

Unauthenticated User Definitions for stand-alone servers
User ID..: H6GUEST      UID...: 5001
Group....: H6GUESTG    UID...: 5502

WebSphere Application Server Asynchronous Administration Task
User ID..: H6ADMSH     UID...: 5002

WebSphere Application Server Servant Group Information
Group....: H6SRG       UID...: 5501
```

Figure 8-7 Security domain definitions

Deployment Manager

```
Cell name (short).....: H6CELL
Cell name (long).....: h6cell

Node name (short).....: H6DM
Node name (long).....: h6dmnode

Server name (short)....: H6DMGR
Server name (long)....: dmgr

Cluster transition name: H6DMGR
Controller Information

  Jobname.....: H6DMGR
  Procedure name: H6MGCR
  User ID.....: H6ACRU
  UID.....: 5003

Servant Information

  Jobname.....: H6DMGRS
  Procedure name: H6MGSR
  User ID.....: H6ASRU
  UID.....: 5004

Daemon job name: H6DEMNA

Procedure name.: H6DEMNA
User ID.....: H6ACRU
UID.....: 5003
```

Figure 8-8 Deployment manager definitions

First empty managed node on SYSA

```
Cell name (short).....: H6BASEA
Cell name (long).....: h6basea

Node name (short).....: H6NODEA
Node name (long).....: h6nodea

Admin asynch operations procedure name: H6ADMSH
Procedure Name Definitions

Controller Information

    Procedure name: H6ACRA
    User ID.....: H6ACRU
    UID.....: 5003

Servant Information

    Procedure name: H6ASRA
    User ID.....: H6ASRU
    UID.....: 5004

Control Region Adjunct

    Procedure name: H6CRAA
    User ID.....: H6ACRU
    UID.....: 5003
Daemon job name: H6DEMN

Procedure name.: H6DEMN
User ID.....: H6ACRU
UID.....: 5003
Node group name.....: DefaultNodeGroup

Node Agent Definitions
    Server name (short)...: H6AGNTA
    Server name (long)....: nodeagent
```

Figure 8-9 Empty managed node definitions on SYSA

NODEB and NODEC

These nodes were defined the same as NODEA with the following exceptions:

- ▶ NODEB (see Figure 8-10):

```
Cell name (short).....: H6BASEB
Cell name (long).....: h6baseb

Node name (short).....: H6NODEB
Node name (long).....: h6nodeb
Node Agent Definitions
  Server name (short)...: H6AGNTB
  Server name (long)....: nodeagent
```

Figure 8-10 Empty managed node definitions on SYSB

- ▶ NODEC (see Figure 8-11):

```
Cell name (short).....: H6BASEC
Cell name (long).....: h6basec

Node name (short).....: H6NODEC
Node name (long).....: h6nodec
Node Agent Definitions
  Server name (short)...: H6AGNTC
  Server name (long)....: nodeagent
```

Figure 8-11 Empty managed node definitions on SYSC

Port numbers

We did not use the default ports; we reserved a range of TCP ports for the High Availability cell. We reserved ports between 39900 and 41000. See Figure 8-12.

	Daemon	Deployment Manager	Node Agent	App Srv #
				01
JMX Soap		39910	39920	39930
Bootstrap/ORB IIOP	39902	39911	39921	39931
ORB SSL	39903	39912	39922	39932
Cell/Node Discovery		39913	39923	
Node Multicast Discovery Port			39924	
High availability manager communication port		39915	39925	39933
Service Integration port				39940
Service Integration secure port				39941
Service Integration MQ Interoperability port				39942
Service Integration MQ Interoperability secure port				39943
JMS Direct				39944
JMS Queued				39945
HTTP		39918		39938
HTTP SSL		39919		39939
Interim Values for Base App Server's Daemon				
ORB IIOP	39900			
ORB SSL	39901			

Figure 8-12 Port assignments

WebSphere procedures

In our configuration, we had separate HFSs for the Application Servers and shared WebSphere procedures. We configured WebSphere to use STEPLIB and in the *WebSphere_ProcsZ*, we defined &SYSCClone as part of the STEPLIB data set name.

Example: H6ASRAZ

Example 8-9 H6ASRAZ procedure

```

/* Output DDs
/*
//CEEDUMP DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSOUT DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSPRINT DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
/*
/*StepLib Setup
/*
//STEPLIB DD DISP=SHR,DSN=BB060&SYSCClone..SBBOLD2
// DD DISP=SHR,DSN=BB060&SYSCClone..SBBLOAD

```

8.2.5 IP name on the Location Service daemon definition

When configuring the Location Service daemon, we used a DVIPA address in the IP Name field in the Daemon definitions; see Figure 8-13.

```
Server Customization (4 of 4)

Specify the following to customize your server, then
to continue.

Location Service Daemon Definitions

Daemon home directory:
  /wasv6config/h6cell/h6dmnode/Daemon

Daemon job name:   H6DEMNA

Procedure name.:  H6DEMNA
User ID.....:   H6ACRU
UID.....:       5003

IP name.....:    HAPLEX1.ITSO.IBM.COM
Port.....:      39902
SSL port.....:   39903

Register daemon with WLM DNS:  N
```

Figure 8-13 Location Service daemon definition

The IP name HAPLEX1.ITSO.IBM.COM is equivalent to 9.1.5.150, which has been defined as a DVIPA address.

Since all the servers in a cluster are clones of each other, it doesn't matter which of the daemons services a request.

8.2.6 Port definitions and assignments

Figure 8-14 shows the port definitions with the system IP addresses and the DVIP.

Deployment Manager ports

We initially defined the Deployment Manager ports as non-DVIPA ports by specifying the Host IP name as wtsc49a.itso.ibm.com. We later changed this definition to a DVIPA address to allow us to move the DM to a different system for High Availability.

dmgr

dmgr

dmgr > Ports

Configure important TCP/IP ports which this server uses for connections.

Preferences

New Delete

Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	wtsc49a.itso.ibm.com	39911	No associated transport
<input type="checkbox"/>	CELL_DISCOVERY_ADDRESS	wtsc49a.itso.ibm.com	39913	No associated transport
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	wtsc49a.itso.ibm.com	39915	View associated transport
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	*	39911	No associated transport
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	*	39912	No associated transport
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	wtsc49a.itso.ibm.com	39910	No associated transport

Total 6

Figure 8-14 Deployment Manager ports assignment

Node agent ports on SYSA

Node agents represent the actual physical node where the application servers are running and are not interchangeable, so they are defined using a non-DVIPA address: wtsc49a.itso.ibm.com.

Preferences				
New Delete				
Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	WTSC49A.ITSO.IBM.COM	39921	No associated transports
<input type="checkbox"/>	CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	9202	No associated transports
<input type="checkbox"/>	CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	9201	No associated transports
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	WTSC49A.ITSO.IBM.COM	39925	View associated transports
<input type="checkbox"/>	DRS_CLIENT_ADDRESS	HAPLEX1.ITSO.IBM.COM	7888	No associated transports
<input type="checkbox"/>	NODE_DISCOVERY_ADDRESS	WTSC49A.ITSO.IBM.COM	39923	No associated transports
<input type="checkbox"/>	NODE_IPV6_MULTICAST_DISCOVERY_ADDRESS	ff01::1	5001	No associated transports
<input type="checkbox"/>	NODE_MULTICAST_DISCOVERY_ADDRESS	232.133.104.73	39924	No associated transports
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	39921	No associated transports
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	39922	No associated transports
<input type="checkbox"/>	SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	9901	No associated transports
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	WTSC49A.ITSO.IBM.COM	39920	No associated transports
Total 12				

Figure 8-15 SYSA node agent ports

Node agent ports on SYSB

Node agents represent the actual physical node where the application servers are running and are not interchangeable, so they are defined using a non-DVIPA address: wtsc49a.itso.ibm.com. See Figure 8-16.

[Node agents](#) > [nodeagent](#) > **Ports**

Configure important TCP/IP ports which this server uses for connections.

☒ Preferences

New Delete

☑ ☐ ⬆️ ⬇️ ⬆️ ⬇️

Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	WTSC50A.ITSO.IBM.COM	39921	No associated transports
<input type="checkbox"/>	CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	9202	No associated transports
<input type="checkbox"/>	CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	9201	No associated transports
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	WTSC50A.ITSO.IBM.COM	39925	View associated transports
<input type="checkbox"/>	DRS_CLIENT_ADDRESS	HAPLEX1.ITSO.IBM.COM	7888	No associated transports
<input type="checkbox"/>	NODE_DISCOVERY_ADDRESS	WTSC50A.ITSO.IBM.COM	39923	No associated transports
<input type="checkbox"/>	NODE_IPV6_MULTICAST_DISCOVERY_ADDRESS	ff01::1	5001	No associated transports
<input type="checkbox"/>	NODE_MULTICAST_DISCOVERY_ADDRESS	232.133.104.73	5000	No associated transports
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	39921	No associated transports
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	39922	No associated transports
<input type="checkbox"/>	SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	9901	No associated transports
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	WTSC50A.ITSO.IBM.COM	39920	No associated transports
Total 12				

Figure 8-16 SYSB Node agent ports

Node agent ports on SYSC

We used different Host IP names for each of the three nodes, but kept the port numbers the same in order to simplify our naming conventions. See Figure 8-17.

Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	WTSC54A.ITSO.IBM.COM	39921	No associated transports
<input type="checkbox"/>	CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	9202	No associated transports
<input type="checkbox"/>	CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	9201	No associated transports
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	WTSC54A.ITSO.IBM.COM	39925	View associated transports
<input type="checkbox"/>	DRS_CLIENT_ADDRESS	HAPLEX1.ITSO.IBM.COM	7888	No associated transports
<input type="checkbox"/>	NODE_DISCOVERY_ADDRESS	WTSC54A.ITSO.IBM.COM	39923	No associated transports
<input type="checkbox"/>	NODE_IPV6_MULTICAST_DISCOVERY_ADDRESS	ff01::1	5001	No associated transports
<input type="checkbox"/>	NODE_MULTICAST_DISCOVERY_ADDRESS	232.133.104.73	5000	No associated transports
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	39921	No associated transports
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	39922	No associated transports
<input type="checkbox"/>	SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	9901	No associated transports
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	WTSC54A.ITSO.IBM.COM	39920	No associated transports

Total 12

Figure 8-17 SYSC node agent ports

8.2.7 Define the first server

After we created an empty managed node on SYSA and added it to the Network Deployment cell, we used the Admin console to define the first server in the cluster; see Figure 8-18. Select **Servers** → **Application servers** → **New**.

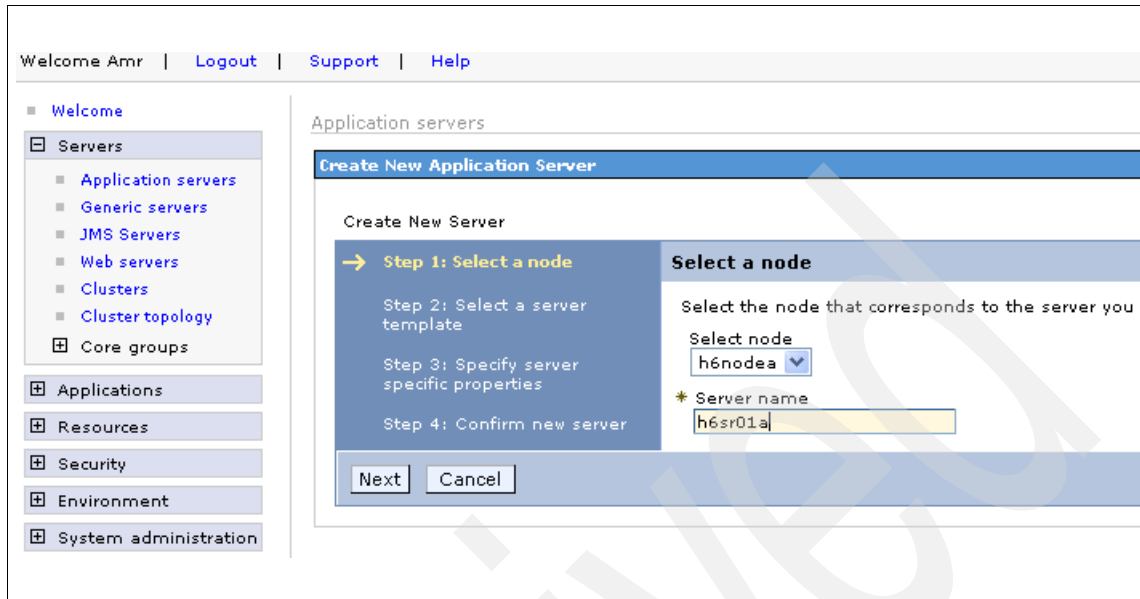


Figure 8-18 Defining a server

Specify name

We entered the following:

- ▶ The node on which this server will run: h6nodea
- ▶ The server name: h6sr01a

Select server template

Application servers

Create New Application Server

Create New Server

Step 1: Select a node

→ **Step 2: Select a server template**

Step 3: Specify server specific properties

Step 4: Confirm new server

Select a server template

Select	Name	Type	Description
<input checked="" type="radio"/>	defaultZOS	System	The WebSphere Default Server Template for z/OS

Previous Next Cancel

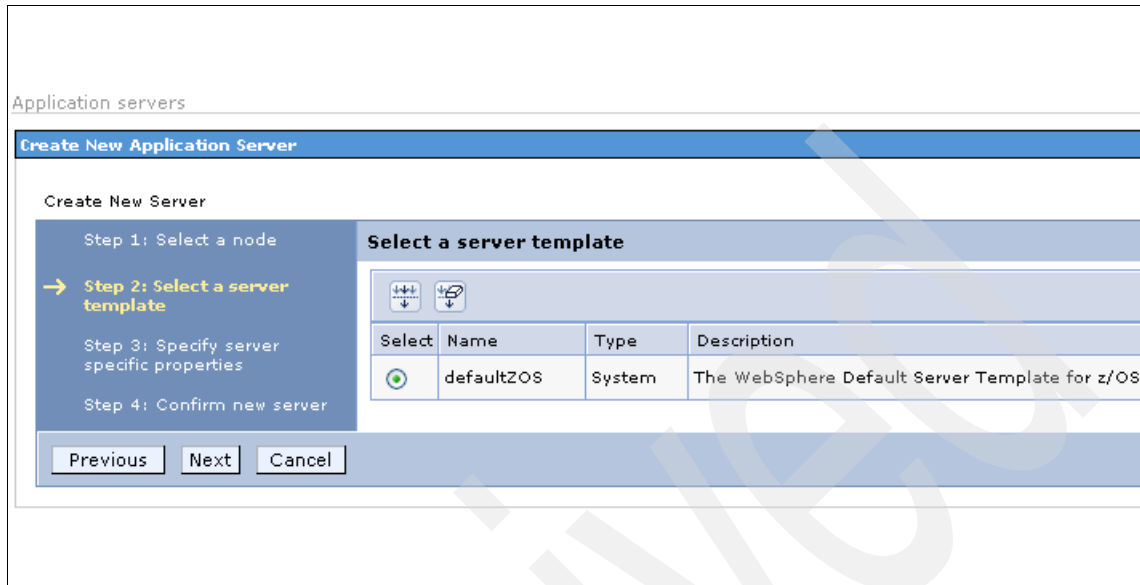


Figure 8-19 Selecting the server template

We chose defaultZOS; see Figure 8-19.

Generate unique HTTP ports

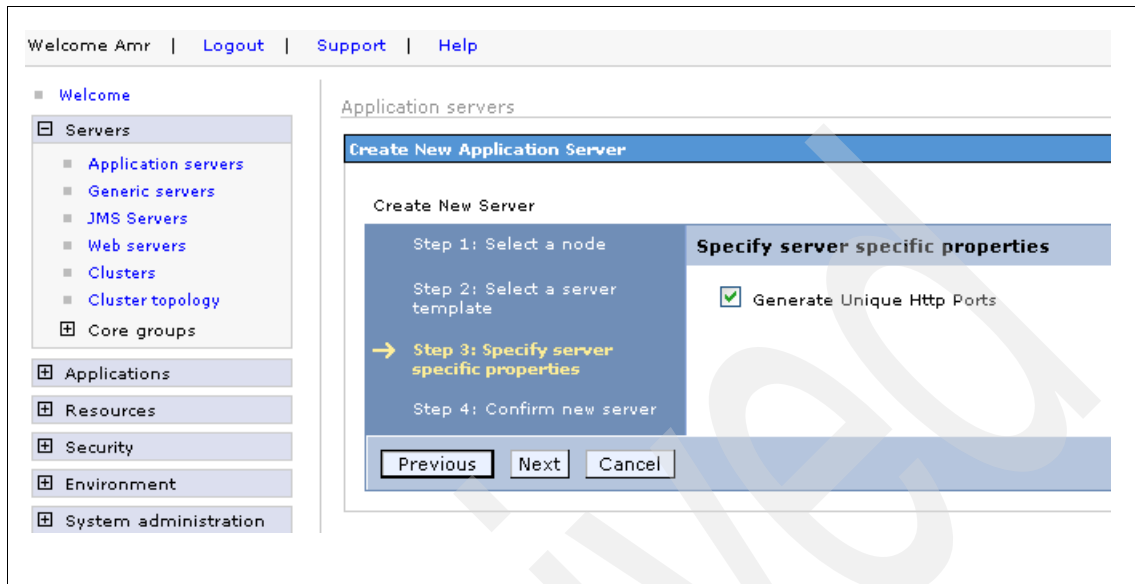


Figure 8-20 Generating unique HTTP ports

Later we remapped these ports according to our convention; see Figure 8-20.

Confirm and save

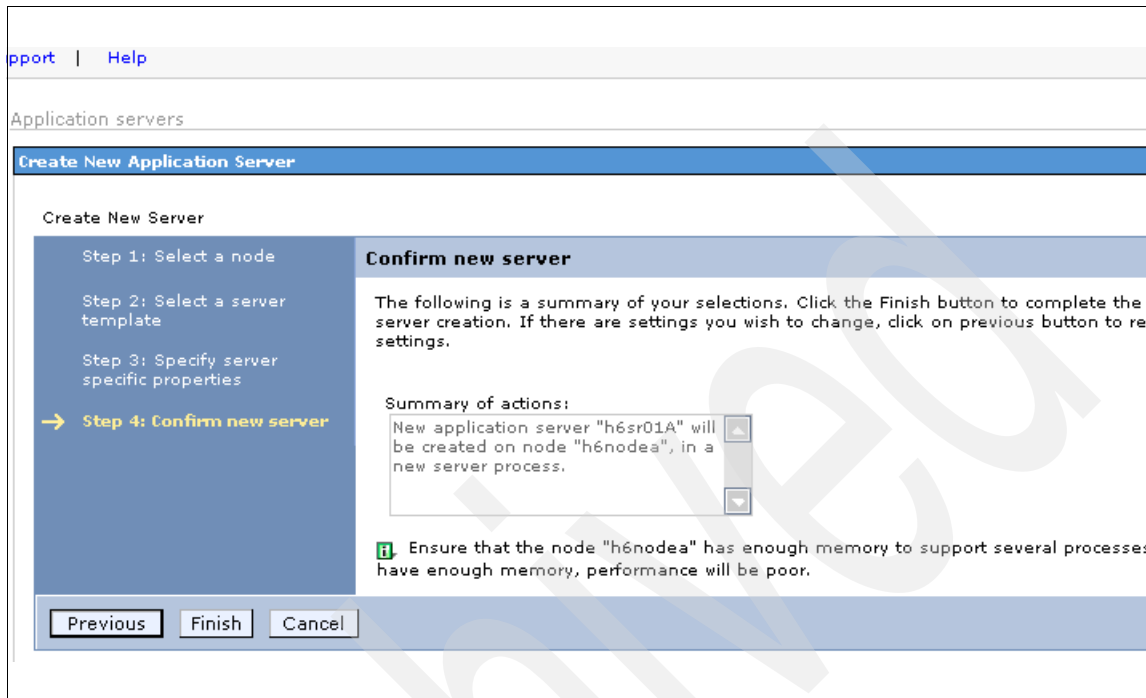


Figure 8-21 Confirming the new server

Finally we saved our new configuration; see Figure 8-21.

8.2.8 Modifying the port settings

After defining the first server, h6sr01a, we went back to change ports to match our design; see Figure 8-22.

Select **Servers** → **Application servers** → **h6sr01a**.

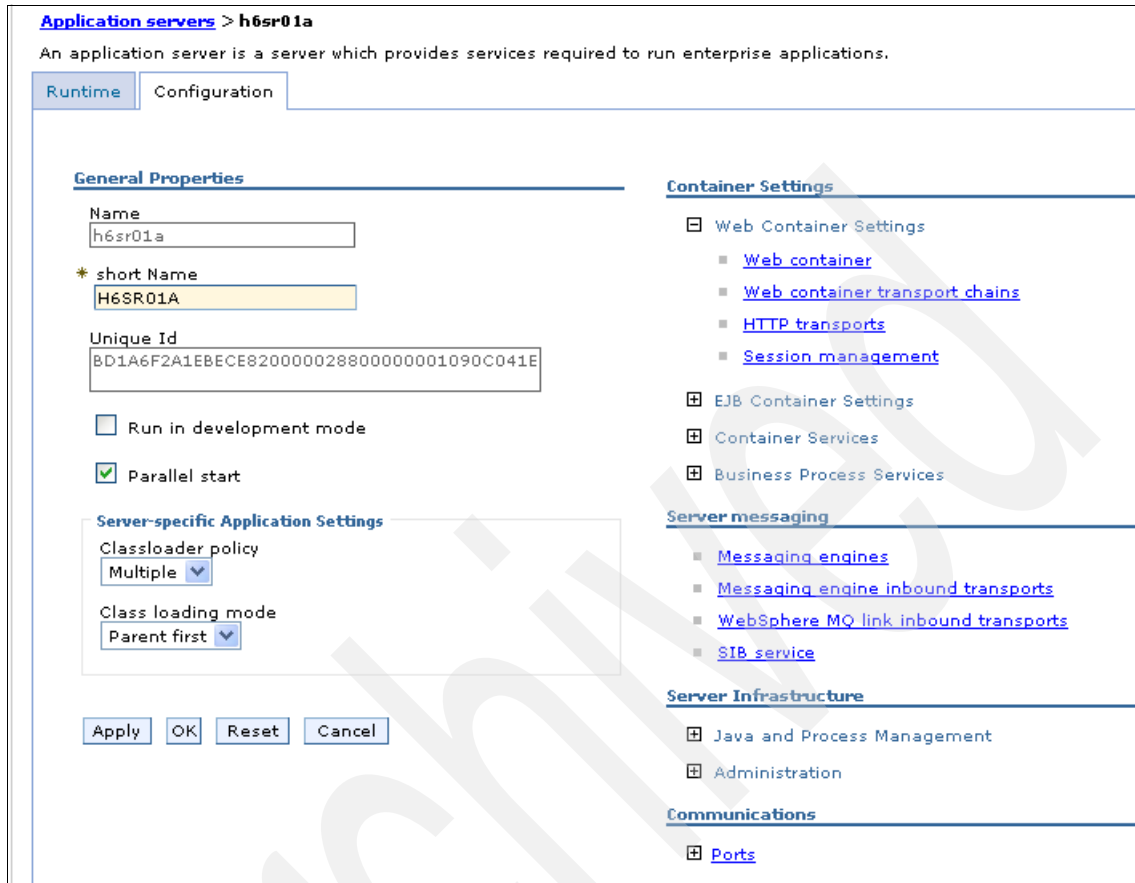


Figure 8-22 Modifying port settings

- ▶ Server Short Name:
BB0C001 -> H6SR01A
- ▶ Cluster Transition Name:
Server Infrastructure → Administration → Custom Properties.
Select ClusterTransitionName → BB0C001 → H6SR01.

Modify the HTTP ports

Application servers > **h6sr01a** > **Ports**

Configure important TCP/IP ports which this server uses for connections.

⊞ Preferences

New Delete

⊞ ⊞ ⊞ ⊞

Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	WTSC49A.ITSO.IBM.COM	39931	No associated transports
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	WTSC49A.ITSO.IBM.COM	39933	View associated transports
<input type="checkbox"/>	JMSSERVER_DIRECT_ADDRESS	HAPLEX1.ITSO.IBM.COM	39944	No associated transports
<input type="checkbox"/>	JMSSERVER_QUEUED_ADDRESS	HAPLEX1.ITSO.IBM.COM	39945	No associated transports
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	39931	No associated transports
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	WTSC49A.ITSO.IBM.COM	39932	No associated transports
<input type="checkbox"/>	SIB_ENDPOINT_ADDRESS	WTSC49A.ITSO.IBM.COM	39940	View associated transports
<input type="checkbox"/>	SIB_ENDPOINT_SECURE_ADDRESS	WTSC49A.ITSO.IBM.COM	39941	View associated transports
<input type="checkbox"/>	SIB_MQ_ENDPOINT_ADDRESS	WTSC49A.ITSO.IBM.COM	39942	View associated transports
<input type="checkbox"/>	SIB_MQ_ENDPOINT_SECURE_ADDRESS	WTSC49A.ITSO.IBM.COM	39943	View associated transports
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	WTSC49A.ITSO.IBM.COM	39930	No associated transports

Total 11

Figure 8-23 Modifying the HTTP ports

8.2.9 New server RACF definitions

The RACF definitions required for the new server are:

- ▶ CBIND class profiles

Access to generic servers:

```
CB.BIND.<domainId>.<cluster> UACC(READ)
PERMIT CB.BIND.<domainID>.<cluster> ID(CFG_groupid)
ACCESS(CONTROL)
```

Access to objects in servers:

```
CB.<domainId>.<cluster> UACC(READ)
```

► SERVER class profiles

Access to controllers using dynamic Application Environments:

```
CB.<server>.<cluster>.<cell> UACC(NONE)
PERMIT CB.<server>.<cluster>.<cell> ID(<SR_userid>
ACCESS(READ)
```

► STARTED class profiles

Controllers

```
<CR_proc>.<CR_jobname> STDATA(USER(CR_userid)
GROUP(CFG_groupid))
<demn_proc>.* STDATA(USER(demn_userid)
GROUP(CFG_groupid))
```

Servants - (started by WLM)

```
<SR_jobname>.<SR_jobname> STDATA(USER(SR_userid)
GROUP(CFG_groupid))
<CRA_jobname>.<CRA_jobname> STDATA(USER(CRA_userid)
GROUP(CFG_groupid))
```

In our definitions we did the following:

1. Defined a standalone server with the dialogs.
2. Generated the jobs.
3. Ran the BBOCBRAJ job.
4. Cut out the resulting commands for the server in the CBIND, SERVER, and STARTED classes.
5. Made them generic using wild-card characters (*).
Example: H6SR01A -> H6SR%%A
6. Modified the BBOCBRAK job to point to the new modified commands, and ran it.

8.2.10 Virtual host alias definitions

The virtual host aliases for the HTTP and the HTTP SSL ports have to be added. You have to define virtual host aliases to the default host defined in plugin.cf.xml:

Environment → **Virtual Hosts** → **default_host** → **Host Aliases**

Figure 8-24 on page 208 shows that the HTTP 39938 and the HTTPS 39939 ports are defined.

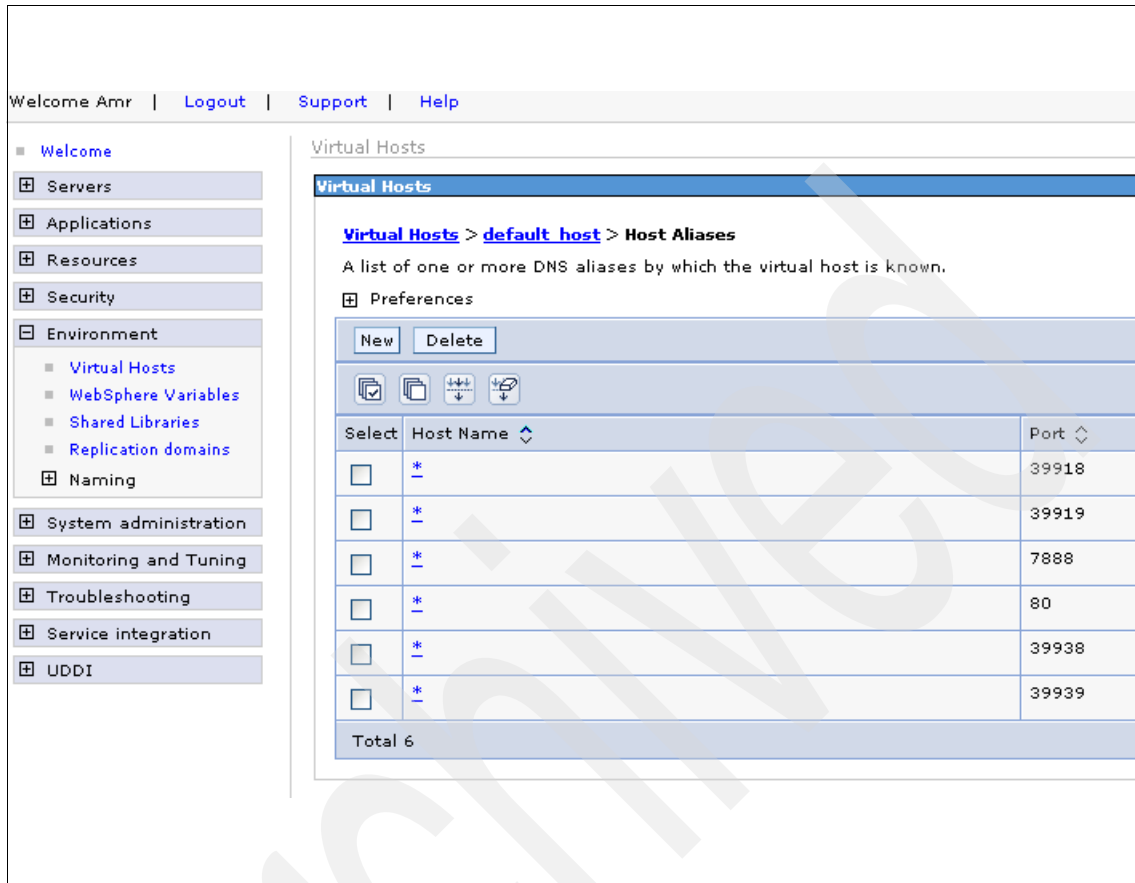


Figure 8-24 Host alias definitions

8.2.11 Defining the cluster

We created empty managed nodes on SC50 and SC54 and added them to the Network Deployment cell. We then used the admin console to define the cluster. The cluster members are h6sr01a, h6sr01b, and h6sr01c on SC49, SC50, and SC54, respectively.

All the cluster members use the same TCP ports.

Note: we also created a replication domain for this cluster. This is one of the steps required to enable a memory-to-memory internal replication domain.

Adding the new cluster

Select **Servers** → **Clusters** → **New**. You will reach the panel shown in Figure 8-25.

Server Cluster

Create a new cluster

Create a new cluster

→ Step 1: Enter basic cluster information.

Step 2: Create cluster members

Step 3: Summary

Enter basic cluster information.

* Cluster name:
h6sr01Cluster

Prefer local

Create a replication domain for this cluster

Existing server:

Do not include an existing server in this cluster

Select an existing server to add to this cluster

Choose a server from this list:
h6cell/h6nodea(6.0.1.1)/h6sr01a

Weight:
2

Next Cancel

Figure 8-25 Creating a cluster - Step 1

Note that the cluster long name is h6r01Cluster, and the h6sr01a server is the starting point.

- ▶ Select **Prefer Local** and **Create a Replication domain for this cluster**.
- ▶ Select **Select an existing server to add to this cluster: h6sr01a**.

In the next panel, shown in Figure 8-26 on page 210, enter these values:

- ▶ The Member name: h6sr01b.
- ▶ The node this cluster member will run on: h6nodeb.
- ▶ Deselect **Generate Unique Http Ports** so that all members of the cluster listen on the same port.

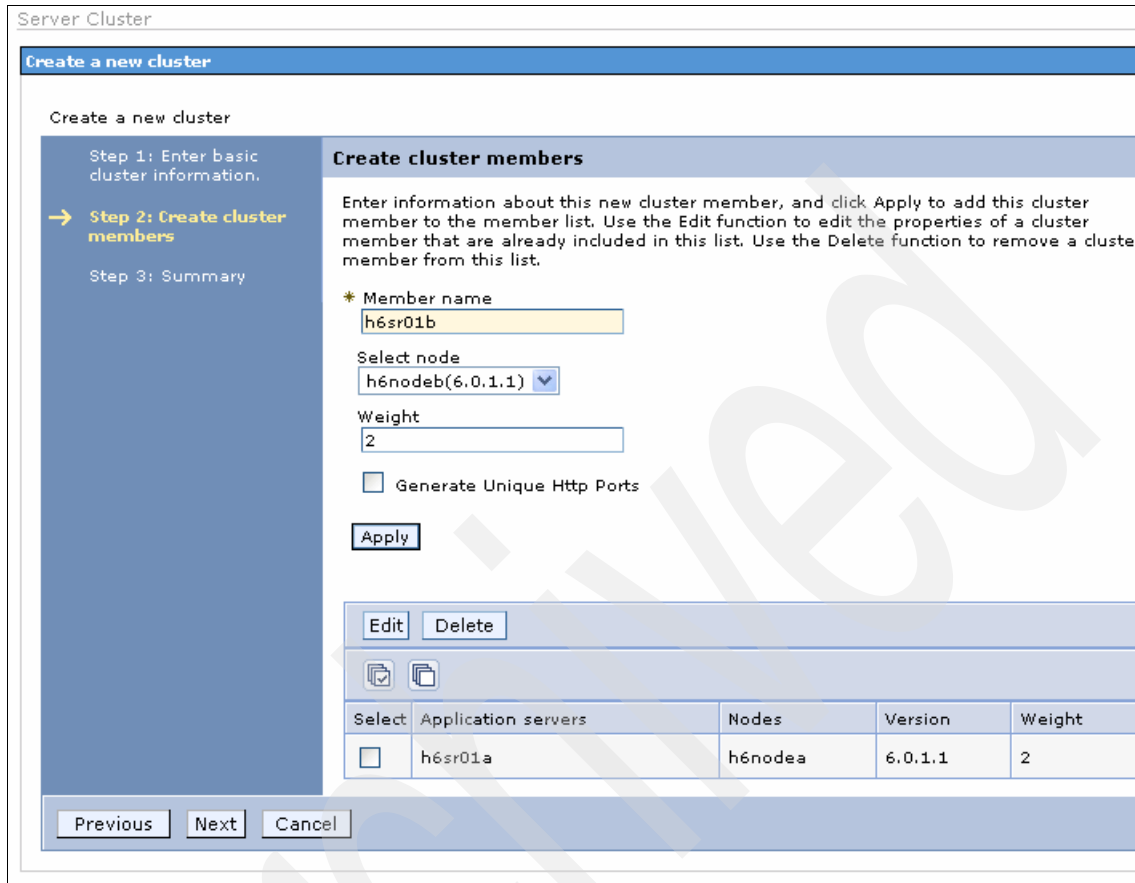


Figure 8-26 Creating a cluster - Step 2

- ▶ Select **Apply**.

On the same panel add the values for the third server, h6sr01c:

- ▶ The Member name: h6sr01c.
- ▶ The node this cluster member will run on: h6nodec.
- ▶ Deselect **Generate Unique Http Ports**.
- ▶ Select **Apply**.

Confirm and save, as shown in Figure 8-27.

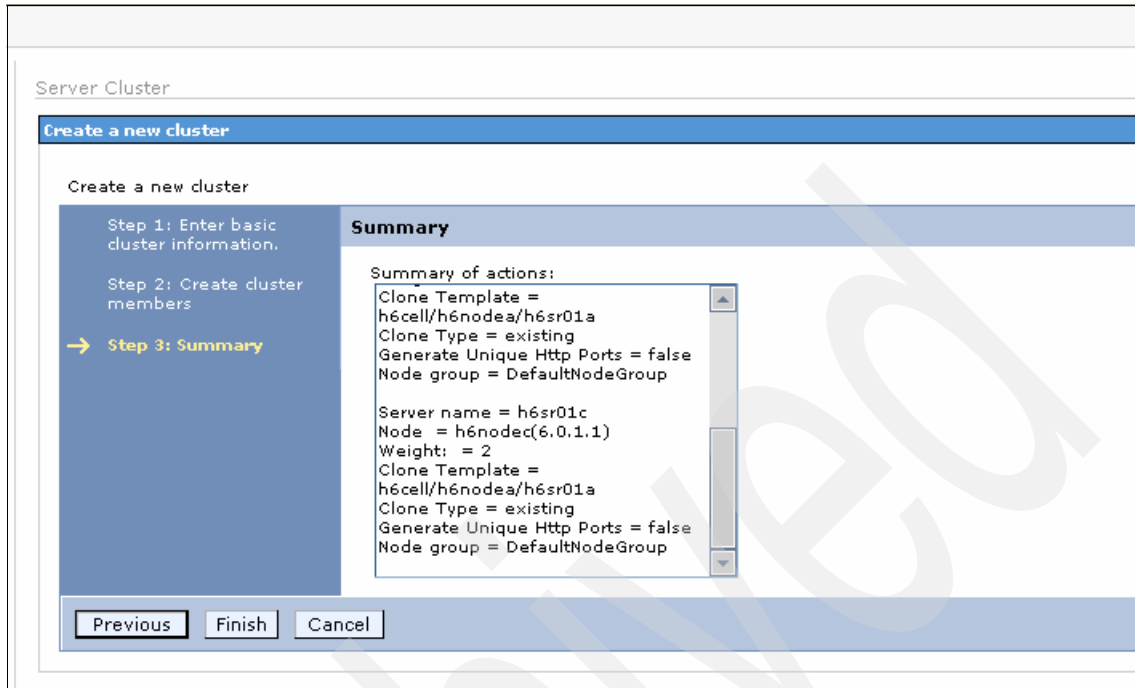


Figure 8-27 Confirm cluster creation

Modify the new clustered server settings and the TCP ports

After creating the second and third servers you have to remap the other TCP ports to be the same as the first server.

We repeated the same steps as in 8.2.8, “Modifying the port settings” on page 204 for the new servers, except the Server Short Names were H6SR01B and H6SR01C. We also checked the RACF definitions for the new servers.

- ▶ Application server h6sr01b ports will be as shown in Figure 8-28.

Application servers > h6sr01b > Ports

Configure important TCP/IP ports which this server uses for connections.

Preferences

New Delete

Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	WTSC50A.ITSO.IBM.COM	39931	No associated transports
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	WTSC50A.ITSO.IBM.COM	39933	View associated transports
<input type="checkbox"/>	JMSSERVER_DIRECT_ADDRESS	HAPLEX1.ITSO.IBM.COM	39944	No associated transports
<input type="checkbox"/>	JMSSERVER_QUEUED_ADDRESS	HAPLEX1.ITSO.IBM.COM	39945	No associated transports
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	39931	No associated transports
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	WTSC50A.ITSO.IBM.COM	39932	No associated transports
<input type="checkbox"/>	SIB_ENDPOINT_ADDRESS	WTSC50A.ITSO.IBM.COM	39940	View associated transports
<input type="checkbox"/>	SIB_ENDPOINT_SECURE_ADDRESS	WTSC50A.ITSO.IBM.COM	39941	View associated transports
<input type="checkbox"/>	SIB_MQ_ENDPOINT_ADDRESS	WTSC50A.ITSO.IBM.COM	39942	View associated transports
<input type="checkbox"/>	SIB_MQ_ENDPOINT_SECURE_ADDRESS	WTSC50A.ITSO.IBM.COM	39943	View associated transports
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	WTSC50A.ITSO.IBM.COM	39930	No associated transports

Total 11

Figure 8-28 h6sr01b port assignments

- ▶ Application server h6sr01c ports will be as shown in Figure 8-29.

Application servers > h6sr01c > Ports

Configure important TCP/IP ports which this server uses for connections.

Preferences

New Delete

Select	Port Name	Host	Port	Transport Details
<input type="checkbox"/>	BOOTSTRAP_ADDRESS	WTSC54A.ITSO.IBM.COM	39931	No associated transports
<input type="checkbox"/>	DCS_UNICAST_ADDRESS	WTSC54A.ITSO.IBM.COM	39933	View associated transports
<input type="checkbox"/>	JMSSERVER_DIRECT_ADDRESS	HAPLEX1.ITSO.IBM.COM	39944	No associated transports
<input type="checkbox"/>	JMSSERVER_QUEUED_ADDRESS	HAPLEX1.ITSO.IBM.COM	39945	No associated transports
<input type="checkbox"/>	ORB_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	39931	No associated transports
<input type="checkbox"/>	ORB_SSL_LISTENER_ADDRESS	WTSC54A.ITSO.IBM.COM	39932	No associated transports
<input type="checkbox"/>	SIB_ENDPOINT_ADDRESS	WTSC54A.ITSO.IBM.COM	39940	View associated transports
<input type="checkbox"/>	SIB_ENDPOINT_SECURE_ADDRESS	WTSC54A.ITSO.IBM.COM	39941	View associated transports
<input type="checkbox"/>	SIB_MQ_ENDPOINT_ADDRESS	WTSC54A.ITSO.IBM.COM	39942	View associated transports
<input type="checkbox"/>	SIB_MQ_ENDPOINT_SECURE_ADDRESS	WTSC54A.ITSO.IBM.COM	39943	View associated transports
<input type="checkbox"/>	SOAP_CONNECTOR_ADDRESS	WTSC54A.ITSO.IBM.COM	39930	No associated transports

Total 11

Figure 8-29 h6sr01c ports assignments

8.2.12 Web container definitions

You have to correlate the definitions of the Web container with the definitions done in the DNS, the plug-in configuration, the Sysplex Distributor, and your application. For an overview at how all definitions have to correlate, refer to 8.2.13, “How these definitions fit together” on page 217.

First you have to define the virtual host that you want to serve on this server. Next you have to enable HTTP session support. We explain these steps in the following sections.

Define the default virtual host

On each server, define the default virtual host, which is the same as in the plug-in xml file. From the admin console, select **Servers** → **Application Servers** → *server name* → **Container Settings** → **Web Container Settings** → **Web container**. See Figure 8-30.

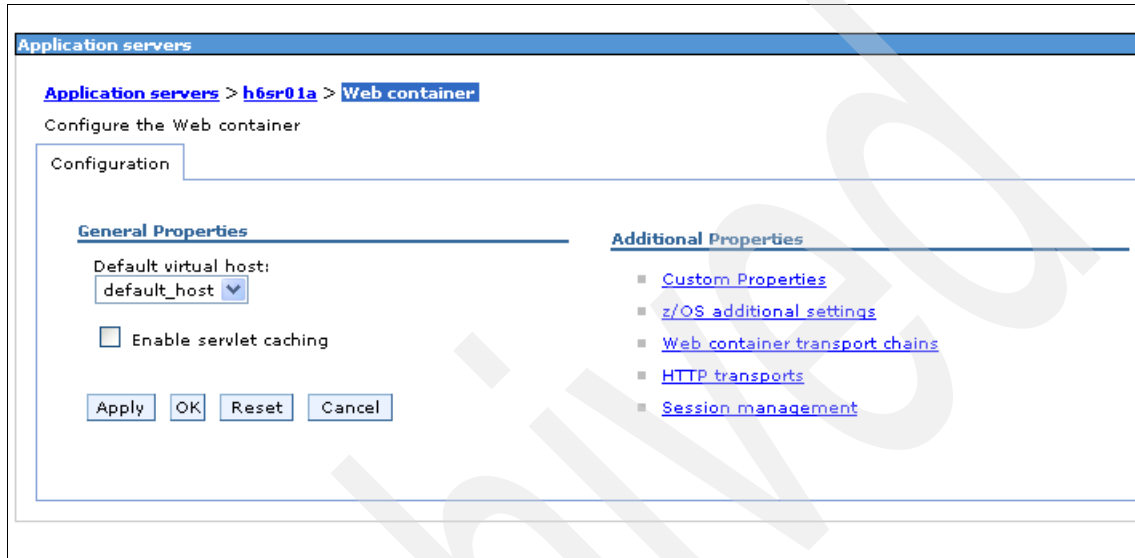


Figure 8-30 Default virtual host definition in the Web container

Enable cookies

Enable cookies on each server by selecting **Servers** → **Application Servers** → *server name* → **Container Settings** → **Session management** → **Enable Cookies**.

We chose to use cookies to implement HTTP sessions. The cookie name *must* be JSESSIONID because the IHS server looks for that cookie to honor session affinity in its workload balancing decisions.

To enable the cookies on h6sr01a, select **Servers** → **Application Servers** → **h6sr01a** → **Container Settings** → **Session management** → **Enable Cookies**.

See Figure 8-31 on page 215.

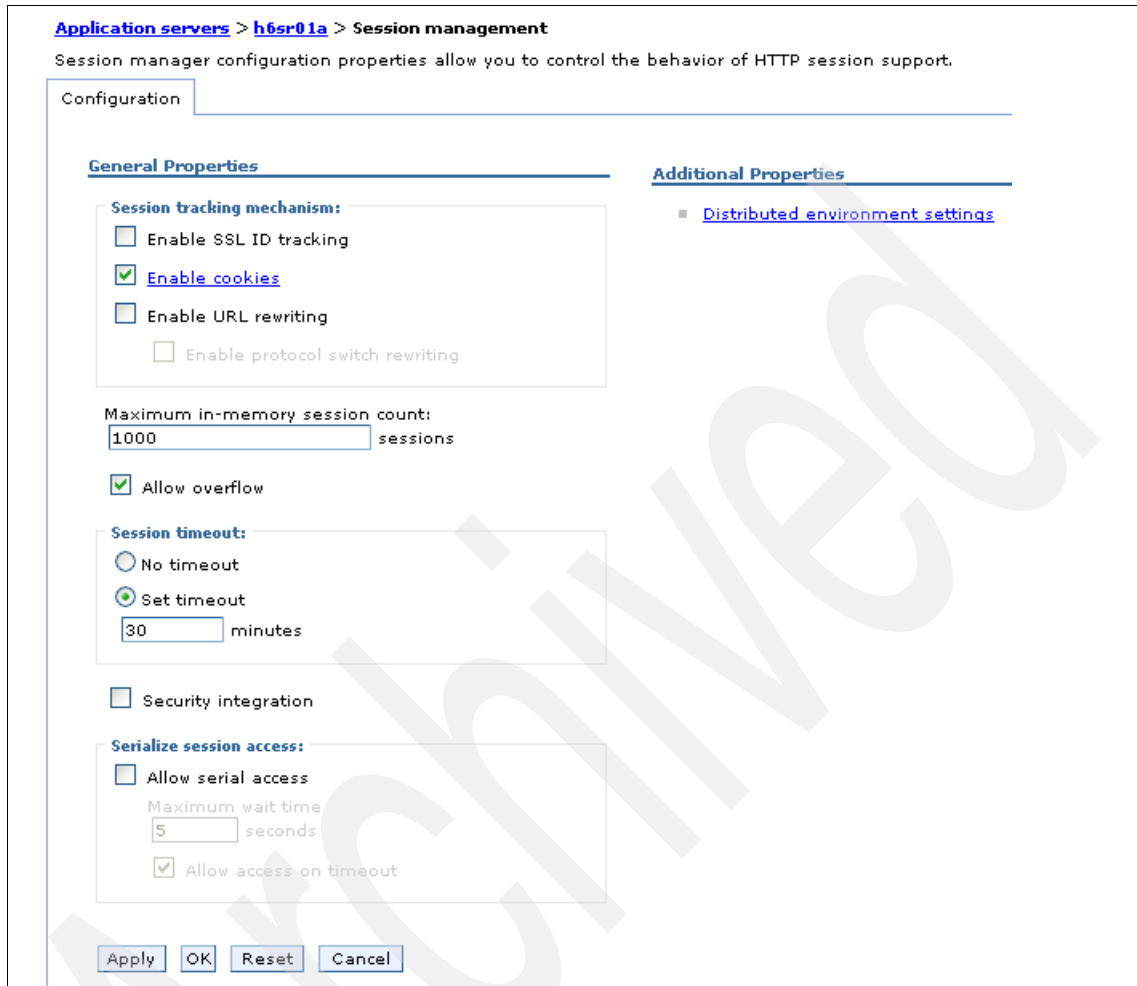


Figure 8-31 Enable Cookies panel

Click **Enable Cookies** on this panel.

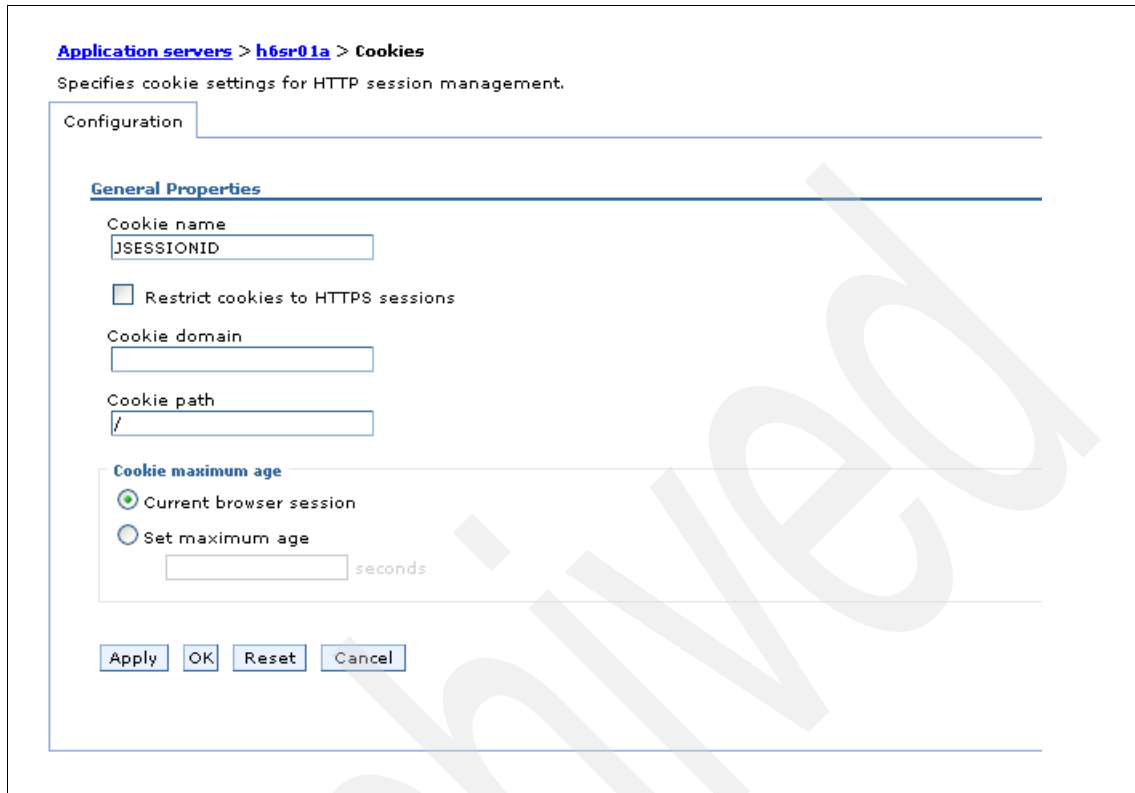


Figure 8-32 Cookie configuration

Repeat the same process for h6sr01b and h6sr01c.

Define CLONE ID

When DRS is defined and active, the server generates, by default, a different CLONE ID instead of the Server Unique Id each time the server is recycled.

This causes the plug-in not to work. To solve this we defined the HttpSessionCloneId value to define a CloneId for every server.

Select **Servers** → **Application Servers** → **server name** → **Container Settings** → **Web Container Settings** → **Web container** → **Custom Properties**.

Select **New** and enter these values (Figure 8-33):

- ▶ Name: HttpSessionCloneId
- ▶ The name of the CloneId for this server: h6sr01aSC49.

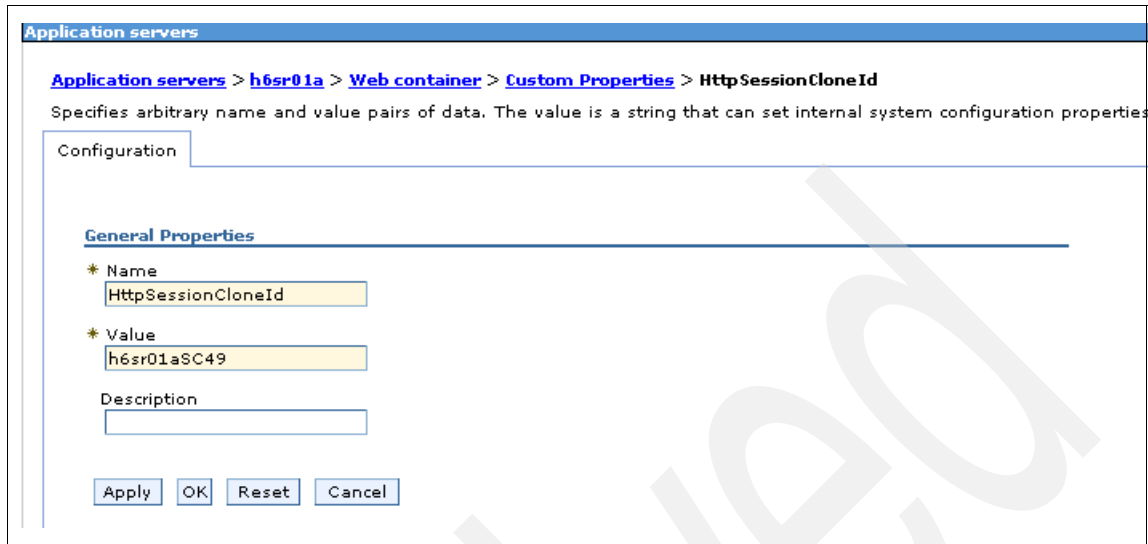


Figure 8-33 Define CloneId for server h6sr01a

Do the same for the other two servers:

- ▶ CloneId for h6sr01b: h6sr01bSC50
- ▶ CloneId for h6sr01c: h6sr01cSC54

8.2.13 How these definitions fit together

In preceding sections, we described the implementation and definition of our WebSphere Edge Server, IHS server, the WebSphere Network Deployment Web server plug-in, Sysplex Distributor and WebSphere application server.

There are many definitions that must correlate, as listed in Table 8-1.

Table 8-1 Correlation of different components

	DNS	Network Dispatcher	IHS Server	plug-in.cfg	Sysplex Distributor	App. Server Instance	application.xml
URL Request Hostname (edgeplex.its o.ibm.com)	Hostname -to-IP address mapping			VirtualHost Name			

	DNS	Network Dispatcher	IHS Server	plug-in.cfg	Sysplex Distributor	Appl. Server Instance	applicati on.xml
IP Address mapped from URL request Hostname (10.1.5.5)	Hostname -to-IP address mapping	Cluster Address	Alias to loopback device				
IP Port of URL Request		Port, defined under the cluster	Port the IHS listen (httpd.conf)	Part of the VirtualHost name			
IP Address of Web server		Server addresses defined under cluster port	Adapter IP addresses				
URI (context root)				URI name			context root
Sysplex Distributor address				Hostname under Cluster Address	VIPADIST RIBUTE DEFINE		
z/OS IP address				Hostname under Server definition			
WebSphere Appl. server HTTP port				Port under Server definition	VIPADIST RIBUTE PORT	BBOC_HTTP_PORT	
WebSphere Appl. server name and instance name				CloneID under server definition		J2EE server name and instance name	

8.3 Other options to set up

In this section we provide information about setting up additional components. First we discuss the new WebSphere feature *internal replication*. Then we discuss setting up session persistence with DB2. Finally we deal with two recovery issues, *peer-to-peer recovery* and *sysplex failure management*.

8.3.1 Setup for memory-to-memory internal replication

One of the mechanisms provided by WebSphere Application Server for z/OS to store HTTP sessions for failover purposes and sharing is called *memory-to-memory internal replication* as explained in 4.2.6, “Memory-to-memory session replication” on page 78 earlier.

The mode we selected for our environment is “Both” and the topology was Peer-to-peer with a local replicator.

Some of the configuration steps were already performed during the cluster creation; see “Adding the new cluster” on page 209.

Verify and configure the internal replication domain

From the WebSphere Administrative console:

- ▶ Select **Environment** → **Replication domains** → **h6sr01Cluster**.
- ▶ You should see a panel similar to the one shown in Figure 8-34 on page 220. We chose **Entire Domain**. Then press **Apply**.

Replication domains

Replication domains > h6sr01Cluster

Specifies the replication properties that all the components of this replication domain use.

Configuration

General Properties

Name: h6sr01Cluster

* Request timeout: 5

Encryption

Encryption type: none

Regenerate encryption key

Number of replicas

Single replica
 Entire Domain
 Specify

Number of replicas:

Apply OK Reset Cancel

Figure 8-34 Replicator entry verification

- ▶ Save and synchronize the configuration.

Verify and configure Session Management

From the WebSphere Administrative console:

- ▶ Select **Server** → **Application Servers**.
- ▶ Select one of the application server members in the cluster: h6sr01a.
- ▶ Navigate to Distributed Environment Settings by selecting **Container Settings** → **Web Container Settings** → **Session Management** → **Distributed Environment Settings**.
- ▶ Check **Memory-to-memory replication** (Figure 8-35) and press **Apply**.

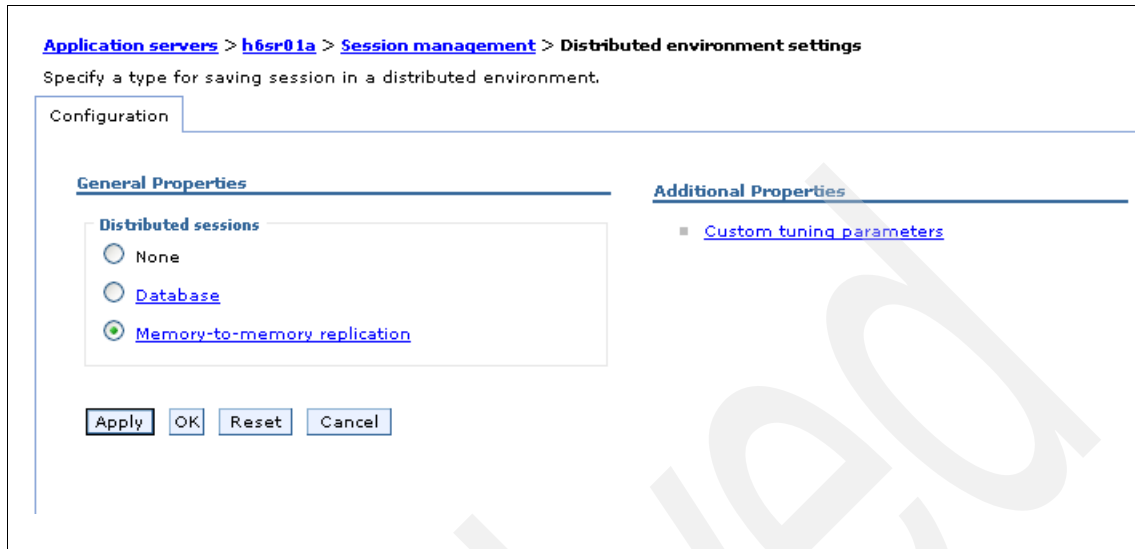


Figure 8-35 Memory-to-memory replication

- ▶ Select **Memory-to-memory Replication** and you should see the panel shown in Figure 8-36.

The replication domain is h6sr01Cluster, Select the Replication mode: **Both client and server**, then press **Apply**.

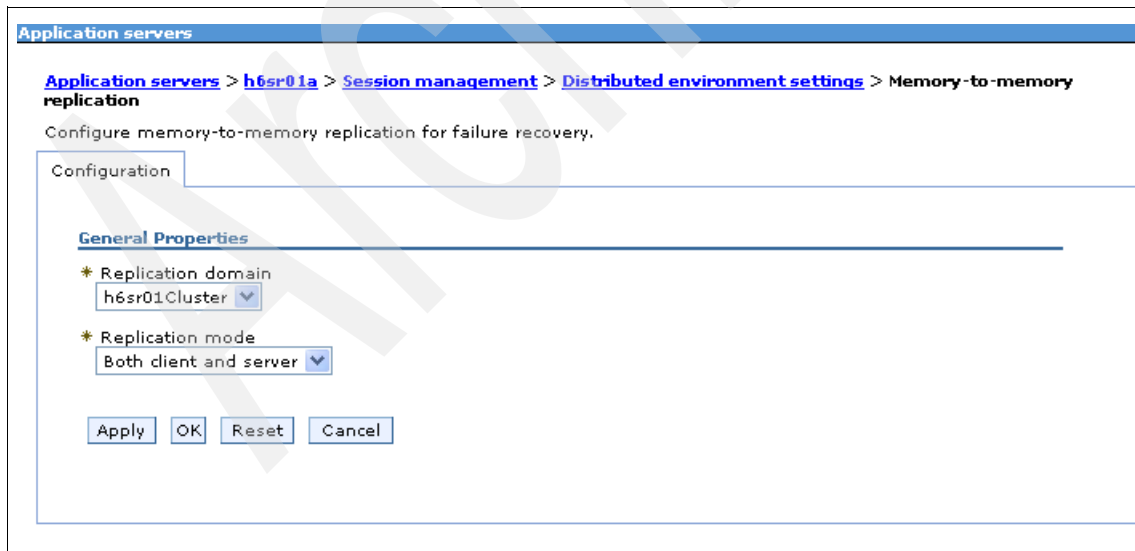


Figure 8-36 Memory-to-memory replication panel

- ▶ Go back to Distributed Environment Settings and select **Custom Tuning Parameters**. You should see the panel shown in Figure 8-37.
For our environment we chose to optimize for failover. Select **Low (optimize for failover)**, then press **Apply**.

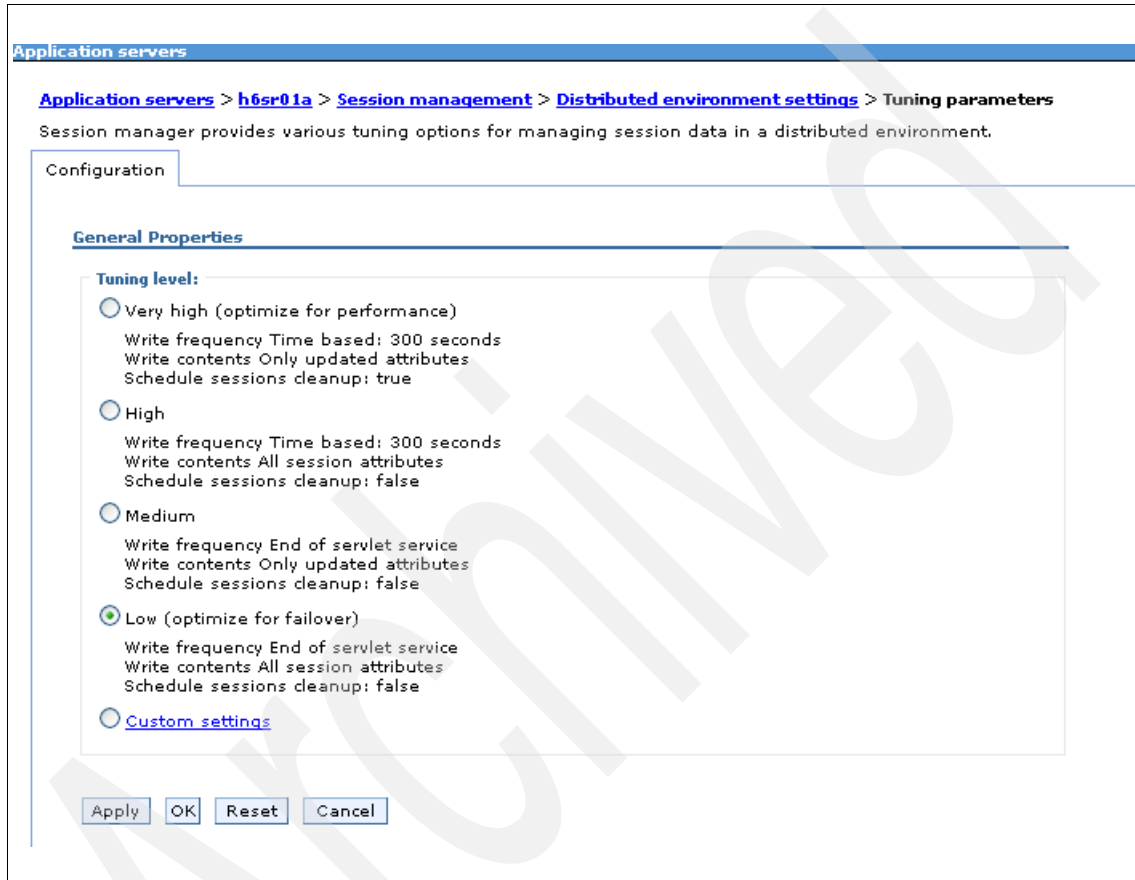


Figure 8-37 Session Manager tuning parameters

- ▶ Repeat these steps for the other application servers in this cluster.
- ▶ Save and synchronize the configuration.

Tip: In our setup, we preferred failover over performance. In many solution designs, however, architects may be forced to trade off some availability for better performance.

For such cases, you may want to set the write frequency to time-based and assign it to 1 second. With this setting, session data will be saved in intervals of one second and will run in a separate thread, thus freeing the application request thread to continue. Be aware, though, that even though the 1 second write frequency still provides good failover capability, 100% failover coverage cannot be guaranteed.

To configure this frequency, go to the Custom settings link, shown in Figure 8-37 on page 222, and change the write frequency to 1.

Verify and configure dynamic cache service

Even though our environment was not configured to take advantage of the dynamic cache services, we found that there is another configurable setting hidden under dynamic cache settings. These settings govern the frequency at which replicators send updates to other replication domain members.

- ▶ From the WebSphere Administrative console, select **Server** → **Application Servers**.
- ▶ Select one of the application server members in our cluster: h6sr01a.
- ▶ Select **Container Settings** → **Container services** → **Dynamic Cache Service**. You should see a panel as shown in Figure 8-38 on page 224.

In the example, the replication domain is h6sr01Cluster. Select:

- Enable service at server startup
- Enable cache replication
- Replication type: **Both push and pull**, and set the push frequency to 0 seconds.

Application servers

Application servers > h6sr01a > Dynamic cache service

The dynamic cache service consolidates caching activities to improve application performance. By caching the response from servlets, Web services, Java Server Pages (JSPs) and WebSphere Application Server commands, the application server does not have to perform the same computations and back-end queries multiple times.

Configuration

General Properties	Additional Properties
<input checked="" type="checkbox"/> Enable service at server startup	<input type="checkbox"/> External cache groups
* Cache size <input type="text" value="2000"/> entries	
* Default priority <input type="text" value="1"/>	
Recovery settings	
<input type="checkbox"/> Enable disk offload Offload location: <input type="text"/>	
<input type="checkbox"/> Flush to disk	
Consistency settings	
<input checked="" type="checkbox"/> Enable cache replication	
Full group replication domain <input type="text" value="h6sr01Cluster"/>	
Replication type <input type="text" value="Both push and pull"/>	
Push frequency <input type="text" value="0"/> seconds	

Figure 8-38 Dynamic cache service setting

- ▶ Repeat these steps for the other application servers in this cluster.
- ▶ Save and synchronize the configuration.

We set up the DRS port as 7888 on all three servers, See Node agent ports in 8.2.6, “Port definitions and assignments” on page 196.

8.3.2 Setup for session persistence with DB2

In our environment, we evaluated availability and failover using memory-to-memory replication as well as persisting HTTP session to the DB2 database.

The setup has changed slightly from earlier versions of WebSphere Application Server for z/OS, and we recommend consulting the WebSphere Application Server InfoCenter for the most recent setup instructions.

In the following sections, we outline the steps to fit our environment.

DB2 datasource setup

- ▶ In order to store the session data of persistent HTTP sessions, you have to create a database, tablespace, and tables. Example 8-10 shows the sample SQL statements we used in our environment.

Example 8-10 Sample SQL statements to define database and tables for session persistence

```
CREATE STOGROUP H6SG09 VOLUMES(TOTDDU) VCAT DB8JU;
```

```
CREATE DATABASE H6HTTP  
STOGROUP SYSDEFLT  
CCSID EBCDIC;
```

```
CREATE TABLESPACE HTTPTS IN H6HTTP  
USING STOGROUP H6SG09  
PRIQTY 512  
SECQTY 1024  
LOCKSIZE ROW  
BUFFERPOOL BP32K;
```

```
CREATE TABLE H6HTTP.HTTPTB (  
  ID          VARCHAR(95) NOT NULL ,  
  PROPID     VARCHAR(95) NOT NULL ,  
  APPNAME    VARCHAR(64) ,  
  LISTENERCNT SMALLINT ,  
  LASTACCESS DECIMAL(19,0),  
  CREATIONTIME DECIMAL(19,0),  
  MAXINACTIVETIME INTEGER ,  
  USERNAME  VARCHAR(256) ,  
  SMALL     VARCHAR(3122) FOR BIT DATA ,  
  MEDIUM   VARCHAR(28869) FOR BIT DATA ,  
  LARGE    BLOB(2097152),  
  SESSROW  ROWID NOT NULL GENERATED ALWAYS
```

```

)
IN H6HTTP.HTTPTS;

CREATE UNIQUE INDEX H6HTTP.HTTPINDEX ON
  H6HTTP.HTTPTB(ID ASC,
  PROPID ASC,
  APPNAME ASC);

CREATE LOB TABLESPACE HTTPSLB IN H6HTTP
  BUFFERPOOL BP32K
  USING STOGROUP H6SG09
  PRIQTY 512
  SECQTY 1024
  LOCKSIZE LOB;

CREATE AUX TABLE H6HTTP.HTTPAXB
  IN H6HTTP.HTTPSLB
  STORES H6HTTP.HTTPTB
  COLUMN LARGE;

CREATE INDEX H6HTTP.HTTPAXINDEX ON
  H6HTTP.HTTPAXB;

GRANT ALL ON H6HTTP.HTTPTB TO H6ASRU;

commit;

```

- ▶ Using the same JDBC provider used for our sample application, we defined the datasource for the HTTP session database. Our JDBC provider in this case is called DB2 390 Local JDBC Provider (RRS).
- ▶ Using the Administrative console, select **Resources** → **JDBC Providers** → **DB2 Universal JDBC Driver Provider (non-XA)** .
- ▶ Add a new datasource under this JDBC provider, for example datasource name HTTP Session Data Source and JNDI jdbc/session.
- ▶ While at the httpSession datasource panel, select **Custom Properties** and make sure that the databaseName property is pointing to the correct value (in our case the value was DB8J, which was the location name).
- ▶ Repeat this for every node participating in the cluster.

Session management changes

WebSphere Application Server provides different levels where you can configure session management. We chose to do it at the Application server level.

Applications installed in these servers use these settings by default. You may change session management settings at the application level if you wish to do so.

The following example shows the location for server h6sr01a. Repeat the procedure for the other members of the cluster.

1. Using the Administrative console, select **Servers** → **Application Servers** → **h6sr01a** → **Web Container** → **Session Management** → **Distributed Environment Settings**. Refer to Figure 8-39 for more details.

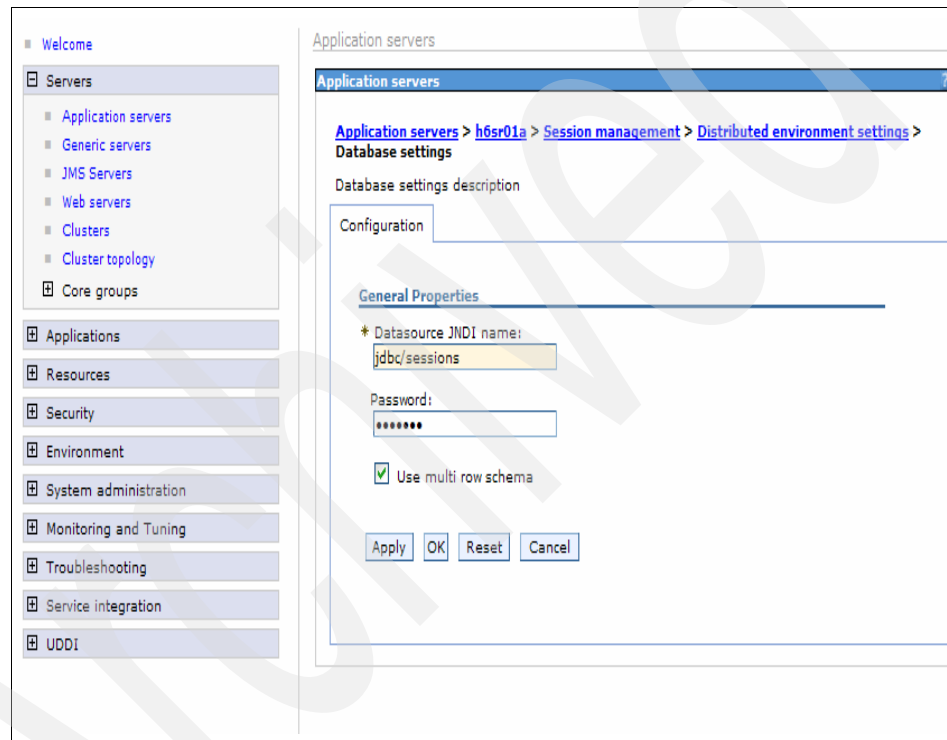


Figure 8-39 Session management database settings

2. Verify that the SessionTableName property for the Web container in each application server is properly set for our environment. Remember to repeat the procedure for all application servers in the cluster.
3. Select **Servers** → **Application Servers** → **h6sr01a** → **Web Container** → **Custom Properties**.

Change the SessionTableName property as shown in Figure 8-40 on page 228.

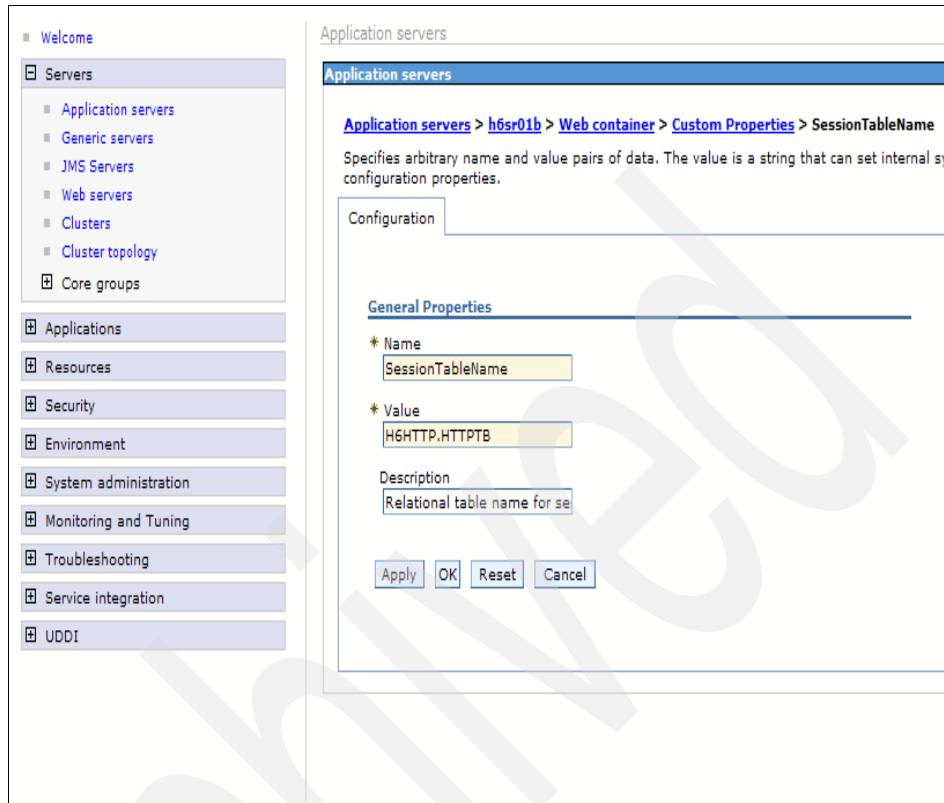


Figure 8-40 Custom property SessionTableName

4. Save and synchronize the configuration, and restart the affected servers.

8.3.3 Peer Restart and Recovery function

Everyone's goal is to provide a highly-available environment, but sometimes it is not possible to avoid unplanned outages. WebSphere Application Server for z/OS has a function called Peer Restart and Recovery (PRR) that allows you to automatically restart the server in a peer system in the sysplex.

PRR can be used in two ways:

- ▶ For application server restart
- ▶ For Deployment Manager restart

Application server restart

When an application server instance fails, it may leave InDoubt and InFlight transactions with unknown results. Transactional managers that were being accessed at that time may still hold locks that were scoped to a transactional unit of recovery.

PRR provides an automated way to recover from this situation by dropping locks that blocked the data and determine the outcomes. The PRR process does not result in lost data.

In order to perform a PRR, the Application server file system must be available on the other systems of the sysplex. The easiest way to do that is to use shared file systems in the affected LPARs.

The PRR process automatically starts when an Application server is restarted in a system different than the one it was configured to run. Our environment is configured to allow PRR to be performed.

Assuming that the original system is still unavailable and that InDoubt and InFlight work has been resolved, new work can be served if this application is deployed in another server present in a system that is available. You cannot run new work with the original server until the system it was configured is brought back online. Restarting the server in a peer system only performs PRR.

Deployment Manager

A very important part of the WebSphere Application server environment running in Network Deployment mode is the Deployment Manager (DMGR). All the administrative functions are performed or driven by the DMGR and synchronized by the Node Agents (NA) in their respective nodes.

Beginning at WebSphere Application Server for z/OS level W502001, PRR was enhanced to allow the DMGR to be restarted in a system other than the one it was originally configured to run in.

In our environment, DMGR was originally configured to run on LPAR SC49.

There are a few configuration prerequisites that we needed to verify or change; Table 8-2 on page 230 lists the details.

Table 8-2 DMGR Peer Restart and Recovery settings

Setting	Host	Port
BOOTSTRAP_ADDRESS	haplex1.itso.ibm.com	39911
CELL_DISCOVERY_ADDRESS	haplex1.itso.ibm.com	39913
ORB_LISTENER_ADDRESS	10.1.6.150	39900
ORB_SSL_LISTENER_ADDRESS	10.1.6.150	39901
SOAP_CONNECTOR_ADDRESS	haplex1.itso.ibm.com	39910

These settings are configured using the Administrative console by selecting **System Administration** → **Deployment Manager** → **End Points**.

The port numbers listed in Table 8-2 have to be defined in your VIPA configuration; refer to 8.1, “Sysplex Distributor” on page 178 for more detailed information about how to change VIPA settings and add these ports.

Note also that the DMGR file system must be shared/accessible across the cell so that it can be reached by other members of a sysplex.

There is no change in the way we access our Administrative console:

<http://haplex1.itso.ibm.com:39918/admin/>

8.3.4 Sysplex Failure Management (SFM)

Sysplex Failure Management (SFM) is an integrated facility of the operating system. It can be used in basic configurations and in Parallel Sysplex configurations.

You can install SFM without having a Coupling Facility (CF), but in order to exploit all the functions that SFM provides, a CF has to be installed. You then have to build and activate an SFM policy.

Note the following points.

SFM should be active in every Parallel Sysplex environment. With SFM active, a failure situation such as the loss of a whole MVS within a Parallel Sysplex, problems with structures, or a loss of connectivity, is managed automatically:

In the case of a structure failure, the SFM will refer to the REBUILDPERCENT parameter of the CFRM policy. A default value of REBUILDPERCENT(100) will result in no automatic rebuild.

In the case of lost connectivity between DB2 and the CF structures, the structures will be rebuilt on an alternate CF that was predefined in the preferences list of the CFRM policy.

Note: In both these cases, there will always be an impact on the applications. Objects will not be available. To maintain high availability, the rebuild percent should be small (for example, 1). In this case, DB2 will always try to rebuild the affected structure.

For a detailed description of SFM, see *z/OS MVS Programming: Sysplex Services Guide*, SA22-7617.

8.4 Maintenance strategy

This section discusses how to install new maintenance levels of WebSphere for z/OS in a sysplex. The goal is to perform this maintenance without disruption to the application workload that is running.

Our environment represents a production environment; we do not refer in this redbook to the maintenance system and the test or integration systems. In a real customer environment, however, there should be a test sysplex consisting of at least two systems and an SMP/E maintenance system.

Note: It is reasonable to have a Parallel Sysplex for testing purposes and not just have a single system. You want your testing environment to be as close as possible to the production environment.

The maintenance system is the only one on which any kind of new software or software maintenance should be installed. From that system, the necessary target libraries and files are distributed to the other systems.

The first step to installing new maintenance is to copy the libraries and the HFS files to the test systems. After successfully running the necessary tests for the new product or maintenance level together with the business applications, the libraries and HFS files will be copied from the maintenance system to the production systems.

To ensure that all systems have the same level of the software, there should be no copying from the test system to the production system; the maintenance system should be the *only* source. This process reflects an assumption we made to ensure that our maintenance strategy works without disruption.

Also note that in this redbook, we discuss maintenance strategies for WebSphere for z/OS to enable high availability. In your environment, however,

you also have to consider the operating system and other products running on your Parallel Sysplex.

8.4.1 The HFS structure for upgrades

Creating the proper HFS structure is a key element to keep in mind when installing new maintenance levels for WebSphere for z/OS without disrupting service to your clients. You use what we call a *rolling upgrade*.

This method allows you to upgrade the WebSphere for z/OS cell by upgrading each node one at a time, keeping the service to clients available while doing the upgrade. The availability of service continues because only one system is removed from the cell, thus allowing the other servers to keep running.

There are different ways of setting up the HFS structure for WebSphere. For detailed information about how to set up an HFS structure for WebSphere, refer to the *WebSphere Application Server Infocenter*.

In our case, we implemented a slightly different HFS structure, as follows:

- ▶ Each system had its own copy of the WebSphere for z/OS product in a unique HFS mounted directly on that system.
- ▶ Since we were running with a shared HFS, we defined a symbolic link to make the HFS accessible:

```
/usr/lpp/zWebSphere/V6R0--> $SYSNAME/zWebSphere.
```

For each upgrade, we simply unmount the current HFS for WebSphere and mount the new HFS at the same point; the symbolic link does not have to be changed over an upgrade. Our HFS structure is shown in Figure 8-41.

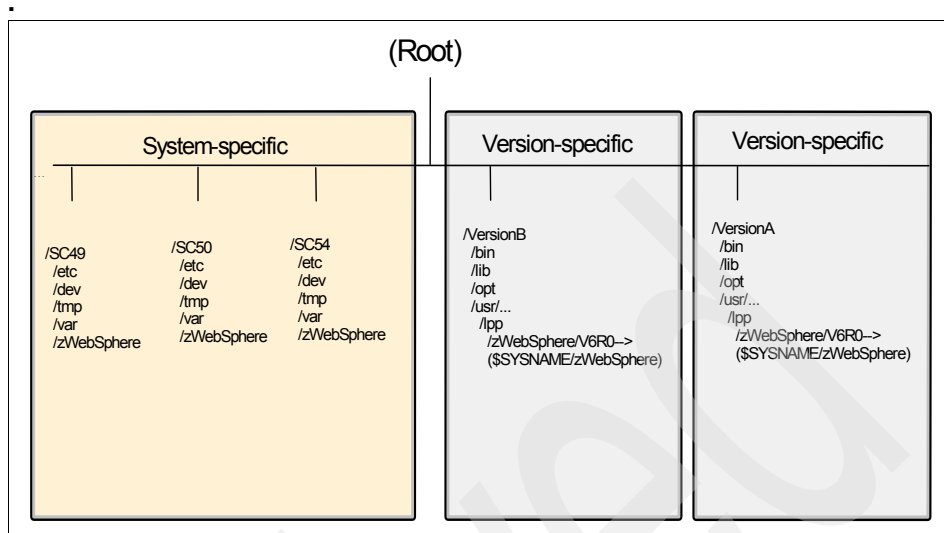


Figure 8-41 WebSphere code HFS structure

8.4.2 Applying maintenance

To upgrade and switch the code level of WebSphere for z/OS throughout the sysplex without disrupting service to your clients, follow these steps:

1. Install the new code level for WebSphere for z/OS.
2. Shut down all application servers and the daemon and node agent on the first system you want to upgrade.
3. Rename the existing WebSphere libraries to a different name.
4. Copy the maintenance libraries and rename them to the original name.
5. Unmount the WebSphere product code HFS and rename it.
6. Dump/Restore the new HFS.
7. Mount the new HFS at the same mountpoint.
8. Load new runtime modules into LPA and update the link list. You can do this dynamically, but it is recommended that you re-IPL the system.

Note: If you are using STEPLIB you don't need to re-IPL.

9. Restart WebSphere for z/OS application servers.
10. Repeat this process for each node in the cell, one at a time.

Tip: In our environment, before enabling new WebSphere Application Server maintenance, we took backups of the file systems holding non-product files (an example of such file systems can be found in Figure 8-5 on page 189).

We found that, if it was necessary to fall back to a previous maintenance level, it was faster and more convenient to restore these backups. Note that you should refer to maintenance documentation for up-to-date recommendations on how to fall back the changes made to the file systems.

8.4.3 HFS versus zFS

The z/OS Distributed File Service zSeries File System (zFS) is a z/OS UNIX System Services (z/OS UNIX) file system that can be used in addition to the hierarchical file system (HFS). zFS file systems contain files and directories that can be accessed with z/OS UNIX application programming interfaces (APIs). These file systems can support access control lists (ACLs).

zFS does not replace HFS, rather zFS is complementary to HFS. HFS is still required for z/OS installation and the root file system must be HFS (unless you are running in a multilevel-secure environment).

In our environment, we used HFS file systems. However, zFS provides a set of improvements that may be considered while deciding which file system to choose. Here are some zFS highlights:

- ▶ Performance

zFS provides significant performance gains in many customer environments accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to HFS.

- ▶ Restart

zFS provides a reduced exposure to loss of updates. zFS writes data blocks asynchronously and does not wait for a sync interval. zFS is a logging file system. It logs metadata updates. If a system failure occurs, zFS replays the log when it comes back up to ensure that the file system is consistent.

- ▶ Space sharing

As an optional function, zFS allows the administrator to define multiple zFS file systems in a single data set. This allows space that becomes available from erasing files in one file system to be available to other file systems in the same data set.

Note: This function is currently only available in non-sysplex sharing environments; it does not apply to our environment.

► Cloning

As an optional function, zFS allows the administrator to make a read-only clone of a file system in the same data set. This clone file system can be made available to users to provide a read-only, point-in-time copy of a file system. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata is copied.

Note: This function is currently only available in non-sysplex sharing environments; it does not apply to our environment.

Results of some performance studies can be found in *UNIX System Services z/OS Version 1 Release 4 Implementation*, SG24-7035. The best results were achieved by using zFS. Note that the better performance is dependent upon how the zFS cache is managed and sized.

Archived

Setting up the messaging infrastructure

Configuring the service integration bus (SIB) for high availability and workload balancing consists of three phases:

- ▶ Configuring the data stores for the messaging engines
- ▶ Configuring the default messaging provider
- ▶ Configuring the core group policies for our messaging engines

Our configuration sets up a push workload balancing scenario as described in 6.5.2, “Push workload balancing” on page 140.

9.1 Set up the data stores

The first phase consisted of setting up all of the data stores for the messaging engines that we required. We had two messaging engines in our cluster; therefore, we required two data stores. The data store tables will be created in DB2 UDB for z/OS.

Note: Because we were using DB2 UDB for z/OS to hold the data store tables, the option to have a messaging engine automatically create the tables is not available. This means that we had to manually create the tables.

9.1.1 Create the data store tables

To manually create the data stores, we first needed the DDL statements to create the tables. We used a tool called sibDDLGenerator to generate the required DDL statements.

In Unix System Services, change to the wasinstall/bin directory and run the commands shown in Example 9-1. These commands should produce two files called h6me000.ddl and h6me001.ddl in the home directory of the logon user id.

Example 9-1 Commands to generate the DDL statements to create the data stores

```
> sibDDLGenerator.sh -system db2 -version 8.1 -platform zos -schema
h6me000 -database h6db000 -storagegroup h6sg000 -statementend \; -user
WASDB2 > ~/h6me000.ddl
```

```
> sibDDLGenerator.sh -system db2 -version 8.1 -platform zos -schema
h6me001 -database h6db001 -storagegroup h6sg001 -statementend \; -user
WASDB2 > ~/h6me001.ddl
```

To ensure that there are no clashes between the two data stores we made sure that the schema name, the database name, and the storage group name were unique for each data store. The user ID, WASDB2, is the user ID that the messaging engines will use to connect to DB2.

Note: For the full syntax of the sibDDLGenerator command, see:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.pmc.zseries.doc\ref\rjm0630_.html

The next step was to copy the files from Unix System Services to a previously allocated Fixed Blocked (FB) 80-byte data set so that we could access the DDL statements via SPUFI.

To do this, enter the commands shown in Example 9-2.

Example 9-2 Commands to copy the DDL statements to a FB80 data set

```
> cp h6me000.ddl “/”pradine.spufi.input(h6me000)”
```

```
> cp h6me001.ddl “/”pradine.spufi.input(h6me001)”
```

These commands will create two new members, H6ME000 and H6ME001, in the specified data set.

The next step was to edit the DDL statements before submitting them. The VCAT parameter of the CREATE STOGROUP statement must be changed to a suitable high-level qualifier for the data sets that will be allocated to hold the tables.

In Figure 9-1 we show that we changed the VCAT to DB8JU on our system.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT PRADINE.SPUFI.INPUT(H6ME000) - 01.02 Columns 00001 00072
Command ==> Scroll ==> PAGE
***** ***** Top of Data *****
==MSG> -Warning- The UNDO command is not available until you change
==MSG> your edit profile using the command RECOVERY ON.
000001 CREATE STOGROUP H6SG000 VOLUMES(TOTDDU) VCAT DB8JU;
000002
000003 CREATE DATABASE H6DB000 STOGROUP H6SG000 BUFFERPOOL BP1;
000004
000005 CREATE TABLESPACE OWNERTS IN H6DB000 USING STOGROUP H6SG000 PRIQTY 3200
000006 SECQTY 320 ERASE NO PCTFREE 0 SEGSIZE 32 BUFFERPOOL BP1 LOCKSIZE TABLE
000007 CLOSE NO;
000008
000009 CREATE TABLE H6ME000.SIBOWNER (
000010 ME_UUID VARCHAR(16),
000011 INC_UUID VARCHAR(16),
000012 VERSION INTEGER,
000013 MIGRATION_VERSION INTEGER
000014 ) IN H6DB000.OWNERTS;
F1=Help F2=Split F3=Exit F5=Rfind F6=Rchange F7=Up
F8=Down F9=Swap F10=Left F11=Right F12=Cancel
```

Figure 9-1 SPUFI edit session

Important: The VCAT parameter must be changed to a suitable high-level qualifier for your system, before submitting the DDL statements, in order to avoid errors.

The DDL statements can be further customized to your system at this time, but this is optional.

An additional change that we made was to change the VOLUMES parameter on the CREATE STOGROUP statement to point to the volume TOTDDU. This can also be seen in Figure 9-1.

Important: Do not modify either the table names, or the column names.

Once we completed editing the DDL statements, we submitted them. All statements should execute cleanly. The expected output is shown in Figure 9-2.

```
Menu Utilities Compilers Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
BROWSE   PRADINE.SPUFI.OUT                               Line 00000358 Col 001 080
Command ==>>>                                         Scroll ==>>> CSR
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
GRANT SELECT,INSERT,UPDATE ON H6ME000.SIBKEYS TO WASDB2;
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 49
DSNE621I NUMBER OF INPUT RECORDS READ IS 217
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 373
***** Bottom of Data *****
  F1=Help   F2=Split  F3=Exit   F5=Rfind  F7=Up     F8=Down   F9=Swap
  F10=Left  F11=Right F12=Cancel
```

Figure 9-2 Expected SPUFI output

Finally, we repeated the steps in SPUFI, described above, in order to create our second data store.

9.1.2 Create the JDBC data source for the messaging engines

In order for our messaging engines to access their data stores, we needed to create at least one JDBC data source. Proceed as follows:

1. Select **Resources** → **JDBC Providers**.
2. Set the scope to the cluster or cell level. We selected the cell level on our system.
3. Click **New**.
 - a. For the database type, select **DB2**.
 - b. For the provider type, select **DB2 Universal JDBC Driver Provider**.
 - c. For the implementation type, select **Connection pool data source**.
4. Click **Next**, and then **Apply**.
5. Select **Data sources**.
6. Click **New**.
 - a. Provide a JNDI name for the data source. On our system we used `jdbc/TraderMEDS`.
 - b. For the Database name we used the location name for the DB2 data sharing group. On our system, this was `DB8J`.
 - c. For the Driver type we used type 4.
 - d. For the Server name we used `HAPLEX1.ITS0.IBM.COM`.
 - e. For the Port number we used 38110.
7. Click **Apply**.
8. Select **J2EE Connector Architecture (J2C) authentication data entries**.
9. Click **New**.
 - a. Select an Alias. On our system we entered `Trader ME alias`.
 - b. Enter a user ID. On our system we entered `wasdb2`. This is the same user ID that we entered for the `-user` parameter of the `sibDDLGenerator` command.
 - c. Enter a Password.
 - d. Click **OK**.

Note: There is no need to specify the alias on the data source, because we will specify it later when we configure the messaging engines.

10. This JDBC provider uses environment variables to locate the jar files for the driver. Ensure that the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable is set correctly.
 - a. Select **Environment** → **WebSphere Variables**.
 - b. Set the scope to the node level, as appropriate. On our system, we have three nodes, h6nodea, h6nodeb, and h6nodec.
 - c. Set the value for the variable DB2UNIVERSAL_JDBC_DRIVER_PATH to the path containing the JDBC driver jars.
11. Save the changes.

Important: The DB2UNIVERSAL_JDBC_DRIVER_PATH must be set appropriately for *every* node.

In our configuration we created just one data source to be used by both messaging engines, but the alternative, to create a separate data source for each messaging engine, would also have been valid.

In the data source configuration we used the sysplex distributor DVIPA address, HAPLEX1.ITSO.IBM.COM. for the DB2 server address. This was to improve the availability of the DB2 subsystem. If one of the members of the DB2 data sharing group were to go down, then the Sysplex Distributor has the ability to route connections to one of the other systems.

This was also the reason that we selected the type 4 driver option, instead of the type 2 driver option, as this allows the messaging engines to connect to any available DB2 subsystem in the data sharing group.

The final change that we made to the DB2 systems was to reduce the DB2 Idle Thread Timeout value to five minutes. In the event of a messaging engine failure, the table lock, previously held by the messaging engine, is released in a timely manner. When the messaging engine is started on another server, it can obtain the table lock and complete the failover.

Important: In order to benefit from the database connections being routed to other DB2 systems by the Sysplex Distributor, any locks that are held by the DB2 system that went down need to be released in a timely manner. To ensure that this occurs, the use of a recovery system such as ARM to restart the DB2 system, in order to release any locks, may be required.

9.2 Configure the service integration bus (SIB)

The next phase was to configure the messaging resources that we required, including: IE, the bus, and the messaging engines.

9.2.1 Create the bus

No service integration buses are defined by default. To create a bus (Figure 9-3):

1. Select **Service integration** → **Buses**.
2. Click **New**.
 - a. Enter a Name. In our configuration we used h6bus.
 - b. Click **OK**.
3. Save the changes.

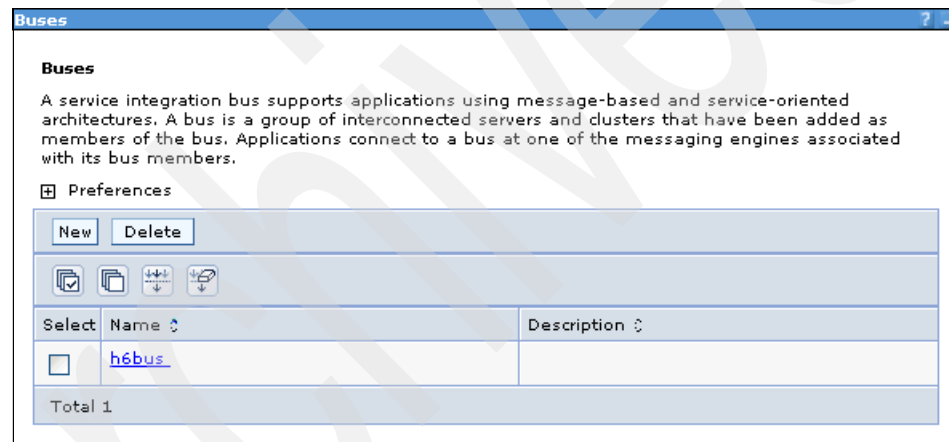


Figure 9-3 A new bus

9.2.2 Add the cluster to the bus

Adding our cluster to the bus will automatically create a messaging engine on the cluster. Here is what we did:

1. Select **Service integration** → **Buses**.
2. Select the bus. On our system, this was called h6Bus.
3. Select **Bus members**.
4. Click **Add**.
 - a. Select the **Cluster**.

- b. Select the cluster from the drop-down menu. On our system, we selected h6sr01C1uster.
 - c. Provide the Data source JNDI name. This is the data source that will allow the messaging engine to access its data store. On our system, this was jdbc/TraderMEDS.
 - d. Click **Next** and then click **Finish**.
5. We had to complete the configuration of the messaging engine before we completed this task. Select **Service integration** → **Buses**.
6. Select the bus. On our system, this was h6Bus.
7. Select **Messaging engines**.
8. Select the messaging engine. On our system, it was h6sr01C1uster.000-h6bus.
9. Select **Data store**.
 - a. Change Schema name to the schema name of the data store for this messaging engine. On our system, this was H6ME000.
 - b. Select the authentication alias from the drop-down list. On our system, this was h6dmnode/Trader ME Alias.
 - c. Untick the **Create tables** option. This option is not supported when using DB2 UDB for z/OS.
 - d. Click **OK**.
10. Save the changes.

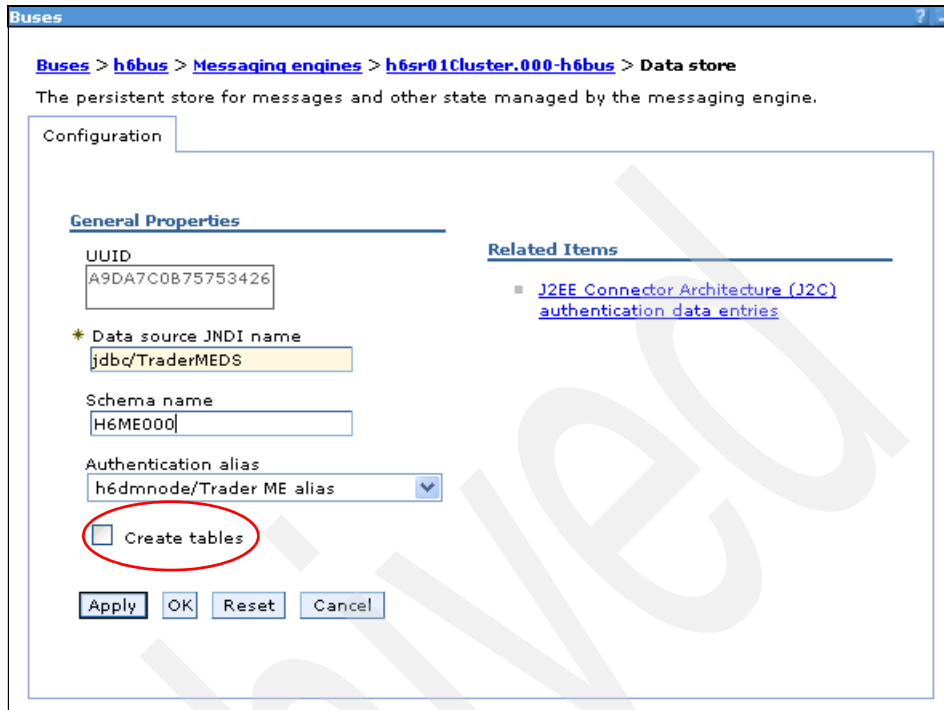


Figure 9-4 h6sr01Cluster.000-h6bus data store configuration

We now have a single messaging engine called h6sr01Cluster.000-h6bus.

9.2.3 Add additional messaging engines

We had to add an additional messaging engine in order to get the configuration that we required. We did the following:

1. Select **Service integration** → **Buses**.
2. Select the bus. On our system, this was h6Bus.
3. Select **Bus members**.
4. Select the cluster member to which we want to add a messaging engine. On our systems, this was Cluster=h6sr01Cluster.
5. Click **Add messaging engine**.
 - a. Change Schema name to the schema name of the data store for this messaging engine. On our system, this was H6ME001.
 - b. Select the authentication alias from the drop-down list. On our system, this was h6dmnode/Trader ME Alias.

- c. Untick the **Create tables** option. This option is not supported when using DB2 UDB for z/OS.
 - d. Click **OK**.
6. Save the changes.

Select **Service integration** → **Buses** → **h6Bus** → **Messaging engines** to see that we now have a second messaging engine, h6sr01Cluster.001-h6bus.

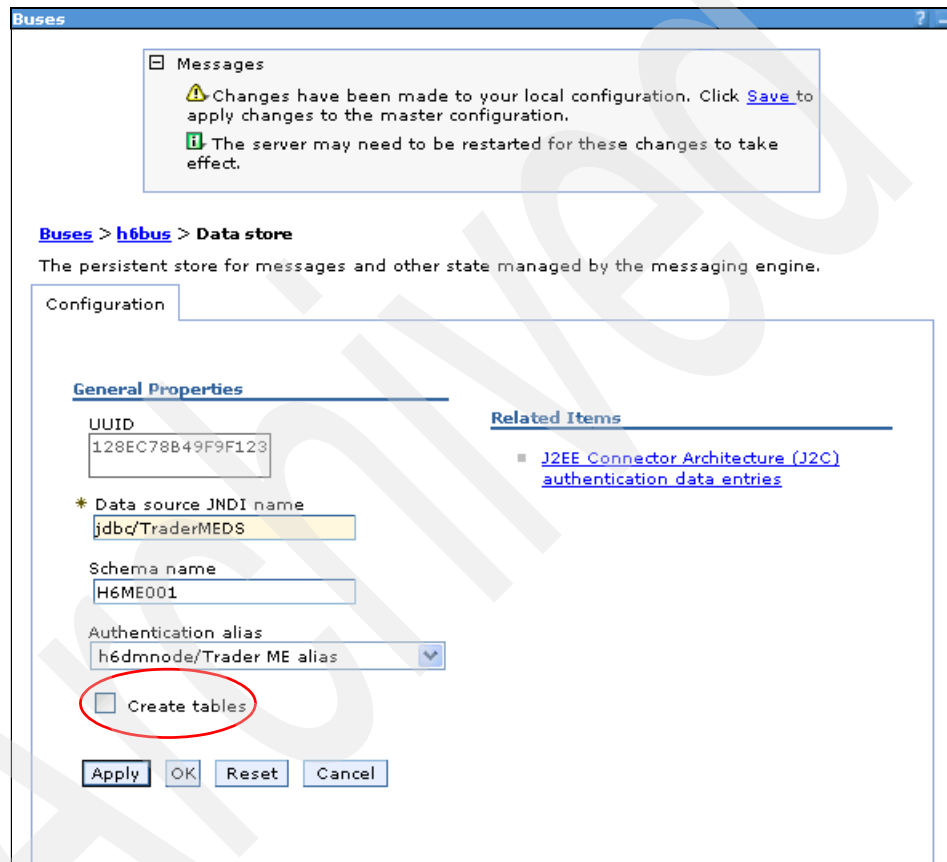


Figure 9-5 h6sr01Cluster.001-h6bus data store configuration

9.3 Configure preferred servers

The final phase is to configure the messaging engines to prefer different servers. We wanted messaging engine h6sr01Cluster.000-h6bus to prefer server h6sr01a (which is on Node A) and messaging engine h6sr01Cluster.001-h6bus

to prefer server h6sr01b (on Node B). Server h6sr01c (on Node C) will be the first alternative server for both messaging engines. See Figure 9-6.

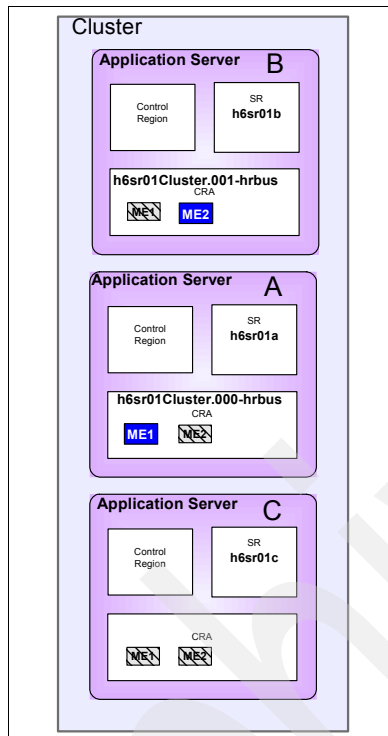


Figure 9-6 Configure MEs for preferred servers

9.3.1 Set up core group policies

To create a core group policy for a messaging engine, do the following:

1. Select **Servers** → **Core groups** → **Core group settings**.
2. Select **DefaultCoreGroup**.
3. Select **Policies**.

Important: There are two policies defined by default, Clustered TM Policy, and Default SIBus Policy. These policies should not be deleted or modified in any way.

4. Click **New**.
 - a. From the drop-down list, select **One of N policy**.

- b. Click **Next**.
5. Provide a name for the new policy. On our systems, we used `h6sr01Cluster.000-h6bus Policy`.
6. Tick the **Fail back** option.
7. Click **Apply**.
8. A warning will show that you must define at least one match criteria. Select **Match criteria**.
9. Click **New**.
 - a. Enter a Name of type.
 - b. Enter a Value of `WSAF_SIB`.
 - c. Click **OK**.
10. Click **New** to define another match criteria.
 - a. Enter a Name of `WSAF_SIB_MESSAGING_ENGINE`.
 - b. Enter a Value of `h6sr01Cluster.000-h6bus`.
 - c. Click **OK**.
11. Return to `h6sr01Cluster.000-h6bus Policy`.
12. Select **Preferred servers**.
13. Select the servers that are to be preferred servers using the Add button. On our systems we selected the following servers:
 - `h6nodea/h6sr01a`
 - `h6nodec/h6sr01c`
 - `h6nodeb/h6sr01b`
 This is shown in Figure 9-7.

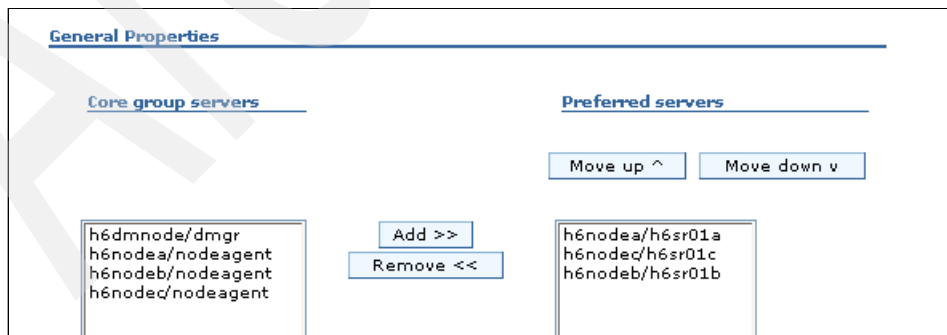


Figure 9-7 Preferred servers for `h6sr01Cluster.000-h6bus`

14. Click **OK** twice.

15. Save your changes.
 16. Click **New**.
 - a. From the drop-down list select **One of N policy**.
 - b. Click **Next**.
 17. Provide a name for the new policy. On our systems, we used h6sr01Cluster.001-h6bus Policy.
 18. Tick the **Fail back** option.
 19. Click **Apply**.
 20. A warning will show that you must define at least one match criteria. Select **Match criteria**.
 21. Click **New**.
 - a. Enter a Name of type.
 - b. Enter a Value of WSAF_SIB.
 - c. Click **OK**.
 22. Click **New** to define another match criteria.
 - a. Enter a Name of WSAF_SIB_MESSAGING_ENGINE.
 - b. Enter a Value of h6sr01Cluster.001-h6bus.
 - c. Click **OK**.
 23. Return to h6sr01Cluster.001-h6bus Policy.
 24. Select **Preferred servers**.
 25. Select the servers that are to be preferred servers using the Add button. On our systems we selected the following servers:
 - h6nodeb/h6sr01b
 - h6nodec/h6sr01c
 - h6nodea/h6sr01a
- This is shown in Figure 9-8.

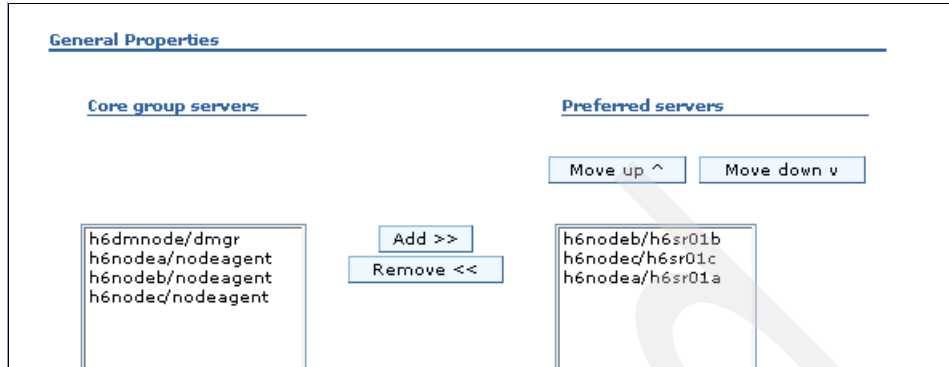


Figure 9-8 Preferred servers for h6sr01Cluster.001-h6bus

26. Click **OK** twice.

27. Save your changes.

The final list of policies is shown in Figure 9-9.

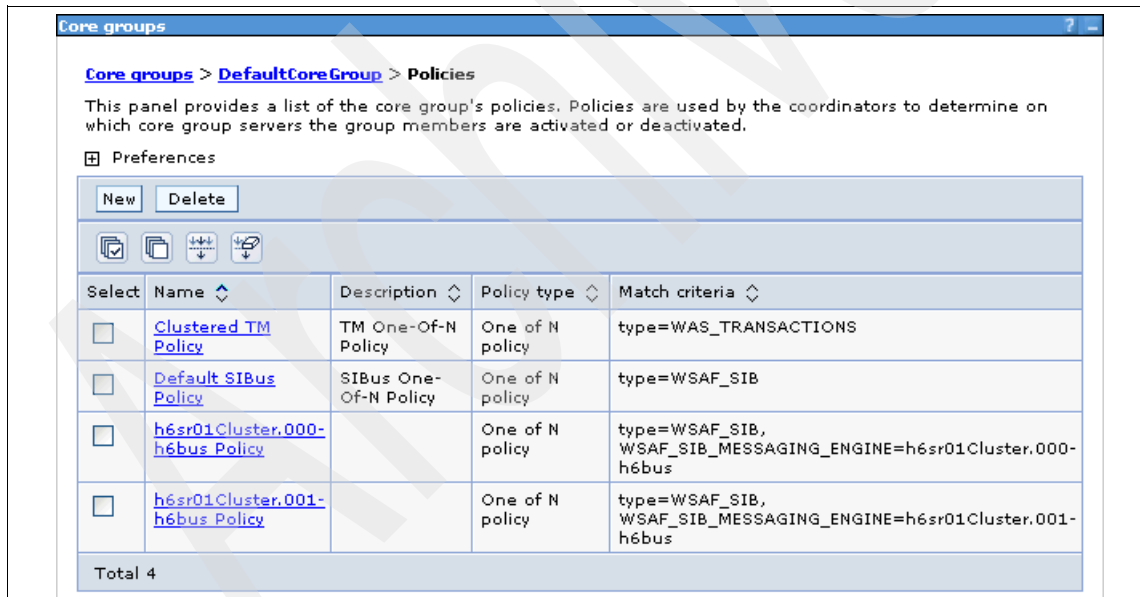


Figure 9-9 New core group policies

9.4 Verify the configuration

Select **Service integration** → **Buses** → **h6Bus** → **Messaging engines** to see that both messaging engines have started. The Status column should indicate whether each messaging engine is started by displaying a green arrow in the row for that messaging engine. This is shown in Figure 9-10.

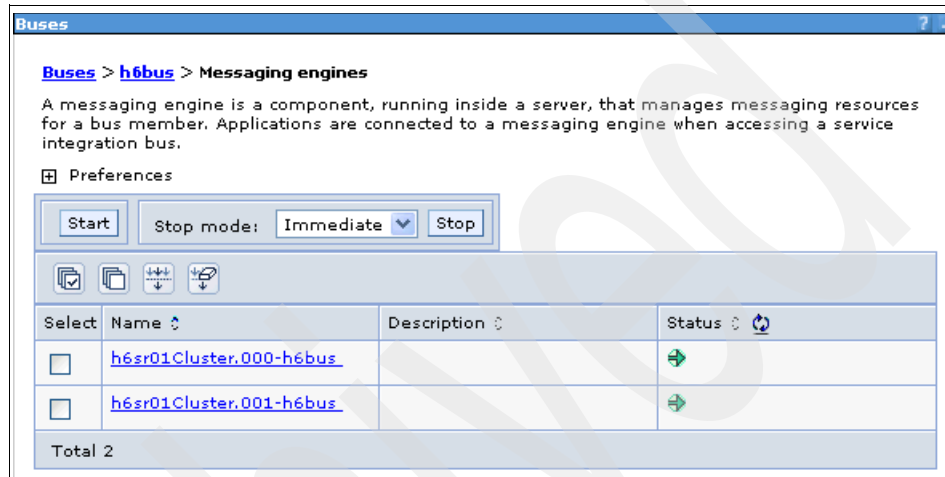


Figure 9-10 Messaging engines in started state

To ensure that the messaging engines are running on the correct servers, you need to check the logs. In WebSphere Application Server for z/OS, this means looking at the joblog for the CRA. The CRA task can easily be identified because it always ends with the letter A.

```

Trace: 2005/06/22 17:42:28.792 01 t=7C5CF0 c=UNK key=P8 (13007002)
  ThreadId: 00000021
  FunctionName: com.ibm.ws.sib.utils.ras.SibMessage
  SourceId: com.ibm.ws.sib.utils.ras.SibMessage
  Category: INFO
  ExtendedMessage: BB000222I: [h6bus:h6sr01cluster.000-h6bus] CWSID0016I:
Messaging engine h6sr01cluster.000-h6bus is in state Started.
Trace: 2005/06/22 17:42:28.792 01 t=7C5CF0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 901
  error message: BB000222I: [h6bus:h6sr01cluster.000-h6bus] CWSID0016I: Messaging
engine h6sr01cluster.000-h6bus is in state Started.

```

Figure 9-11 CRA joblog for server h6sr01a

Figure 9-11 shows the joblog for the CRA in server h6sr01a. On our system, the task was called H6SR01AA. It shows that messaging engine h6sr01cluster.000-h6bus is started in server h6sr01a.

Figure 9-12 shows the joblog for the CRA in server h6sr01b. On our system, the task was called H6SR01BA. It shows that messaging engine h6sr01cluster.001-h6bus is started in server h6sr01b.

```

Trace: 2005/06/22 17:55:43.745 01 t=7CDCF0 c=UNK key=P8 (13007002)
  ThreadId: 00000023
  FunctionName: com.ibm.ws.sib.utils.ras.SibMessage
  SourceId: com.ibm.ws.sib.utils.ras.SibMessage
  Category: INFO
  ExtendedMessage: BB000222I: [h6bus:h6sr01cluster.001-h6bus] CWSID0016I:
Messaging engine h6sr01cluster.001-h6bus is in state Started.
Trace: 2005/06/22 17:55:43.745 01 t=7CDCF0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 901
  error message: BB000222I: [h6bus:h6sr01cluster.001-h6bus] CWSID0016I: Messaging
engine h6sr01cluster.001-h6bus is in state Started.

```

Figure 9-12 CRA joblog for server h6sr01b

Figure 9-13 shows the joblog for the CRA in server h6sr01c. On our system, the task was called H6SR01CA. It shows that messaging engines h6sr01Cluster.000-h6bus and h6sr01Cluster.001-h6bus are both in the joined state. This means that both messaging engines can failover to this server in the event of a failure. h6sr01Cluster.000-h6bus should also be joined to server h6sr01b, and h6sr01Cluster.001-h6bus should also be joined to server h6sr01a.

```
Trace: 2005/06/22 18:00:06.736 01 t=7CB0D0 c=UNK key=P8 (13007002)
  ThreadId: 00000021
  FunctionName: com.ibm.ws.sib.utils.ras.SibMessage
  SourceId: com.ibm.ws.sib.utils.ras.SibMessage
  Category: INFO
  ExtendedMessage: BB000222I: [h6bus:h6sr01Cluster.000-h6bus] CWSID0016I:
Messaging engine h6sr01Cluster.000-h6bus is in state Joined.
Trace: 2005/06/22 18:00:06.736 01 t=7CB0D0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 901
  error message: BB000222I: [h6bus:h6sr01Cluster.000-h6bus] CWSID0016I: Messaging
engine h6sr01Cluster.000-h6bus is in state Joined.
Trace: 2005/06/22 18:00:06.747 01 t=7CB0D0 c=UNK key=P8 (13007002)
  ThreadId: 00000021
  FunctionName: com.ibm.ws.sib.utils.ras.SibMessage
  SourceId: com.ibm.ws.sib.utils.ras.SibMessage
  Category: INFO
  ExtendedMessage: BB000222I: [h6bus:h6sr01Cluster.001-h6bus] CWSID0016I:
Messaging engine h6sr01Cluster.001-h6bus is in state Joined.
Trace: 2005/06/22 18:00:06.747 01 t=7CB0D0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 901
  error message: BB000222I: [h6bus:h6sr01Cluster.001-h6bus] CWSID0016I: Messaging
engine h6sr01Cluster.001-h6bus is in state Joined.
```

Figure 9-13 CRA joblog for server h6sr01c

Archived

Availability tests

In this part we describe the applications we used for our tests and how we tested the infrastructure to make sure it withstood failures in the various components.

We did not test all the situations that we tested in the first version of this redbook. For this update, we did not need to test the hardware failures (CPU, LPAR, and CF), or some of the software failures (LDAP, DB2, and RRS) that had nothing to do with WebSphere, since those components had not changed since the previous edition.

Archived

Applications used to drive the workload

In this chapter we introduce the workload that we ran during our tests, and the applications that were used to drive the workload.

We used applications with the following characteristics:

- ▶ They ran as EJBs.
- ▶ They accessed DB2.
- ▶ They created session objects.

10.1 Applications used to create the workload

The Trader application satisfied all these requirements. This application was developed by IBM/ITSO to model an electronic brokerage service. The Trader application models an online brokerage application, providing transactions such as login, buy, sell, and get quotes.

We used two versions of the application:

- ▶ TraderDB2

This version provides a real world application mix of servlets, JSPs, entity beans, and *session* beans. The session bean uses JDBC, or an entity bean, to read and update a database of brokerage accounts.

- ▶ TraderJMS

This version provides a real world application mix of servlets, JSPs, entity beans, and *message-driven* beans. The message-driven beans also use JDBC, or an entity bean, to read and update a database of brokerage accounts.

For more information about installing the TraderDB2 application, refer to *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064.

10.1.1 TraderDB2

The TraderDB2 application uses JDBC and single-phase commit to read and write data to a database. It uses a servlet to drive a stateless session bean to call JDBC to read or write DB2 data. The EJB then returns the data to be formatted by a JSP.

Table 10-1 lists the functions of the TraderDB2 application.

Table 10-1 *TraderDB2 functions*

Function	Description	Complexity	DB access
Login	User sign-in, session creation	Servlet, JSP, Session EJB	
Logout	User sign-off, session destroy	Servlet, JSP, Session EJB, CMP Bean Update	
Quote	View your current stock holdings	DB2 Connect	Read

Function	Description	Complexity	DB access
Buy	Purchase a number of shares	Servlet, JSP, Session EJB, Read/Update	Read, Update
Sell	Sell a number of shares	Servlet, JSP, Session EJB, Read/Update	Read, Update

10.1.2 TraderJMS

The TraderJMS application is modeled on the TraderDB2 application, so they are very similar. The difference is that the servlet uses JMS to send requests to a message-driven bean instead of invoking a method on a session bean.

The message-driven bean then uses JDBC to read and update data in DB2. It sends a reply to a temporary destination that the servlet is listening to.

Table 10-2 lists the functions of the TraderJMS application.

Table 10-2 TraderJMS function

Function	Description	Complexity	DB access
Login	User sign-in, session creation	Servlet, JSP	
Logout	User sign-off, session destroy	Servlet, JSP	
Quote	View your current stock holdings	DB2 Connect	Read
Buy	Purchase a number of shares	Servlet, JSP, Message-driven EJB, Read/Update	Read, Update
Sell	Sell a number of shares	Servlet, JSP, Message-driven EJB, Read/Update	Read, Update

10.2 WebSphere Workload Simulator

We used WebSphere Workload Simulator to drive the Trader application to create a workload sufficient to load the system and test the high availability aspects of the architecture we developed.

WebSphere Workload Simulator is a tool which allows you to capture the output of interactions between a Web browser and a server, and then replay those same interactions later.

To create the workload for our tests, we captured the interactions between the browser and WebSphere while performing functions from the TraderDB2 and TraderJMS applications.

WebSphere Workload Simulator contains two main components, the controller and the engine, which are described as follows:

- ▶ The controller
This component runs on a workstation and is used to capture and modify the test script. Test scripts are FTPed to the engine machine for execution there. During execution, the controller connects to port 3000 on the engine machine to display a monitor of the test progress.
- ▶ The engine
This component runs on a driving system and executes the script to drive the workload. In our implementation of WebSphere Workload Simulator, we ran the engine in UNIX System Services on z/OS.

Figure 10-1 on page 261 provides an overview of the WebSphere Studio Workload Simulator.

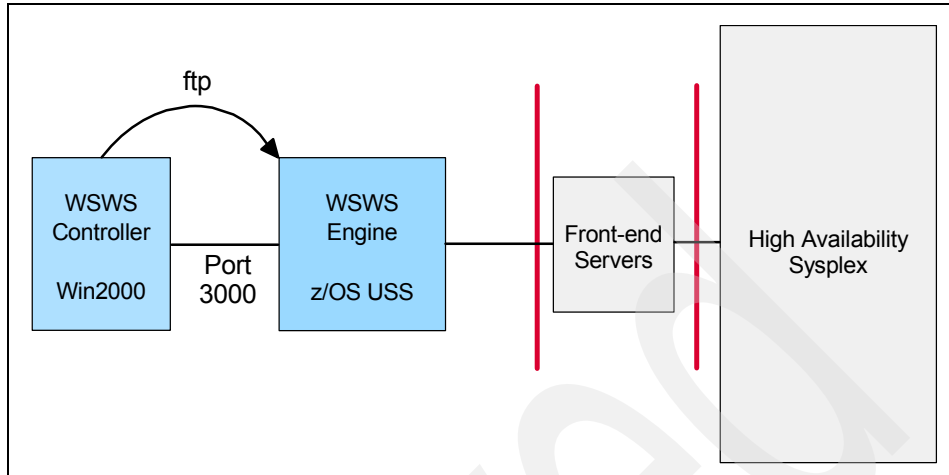


Figure 10-1 WebSphere Workload Simulator overview

When running WebSphere Workload Simulator, the controller communicates with the engine (the driving system) that is actually running the tests. The controller presents a window that allows you to monitor the test progress; refer to Figure 10-2 on page 262.

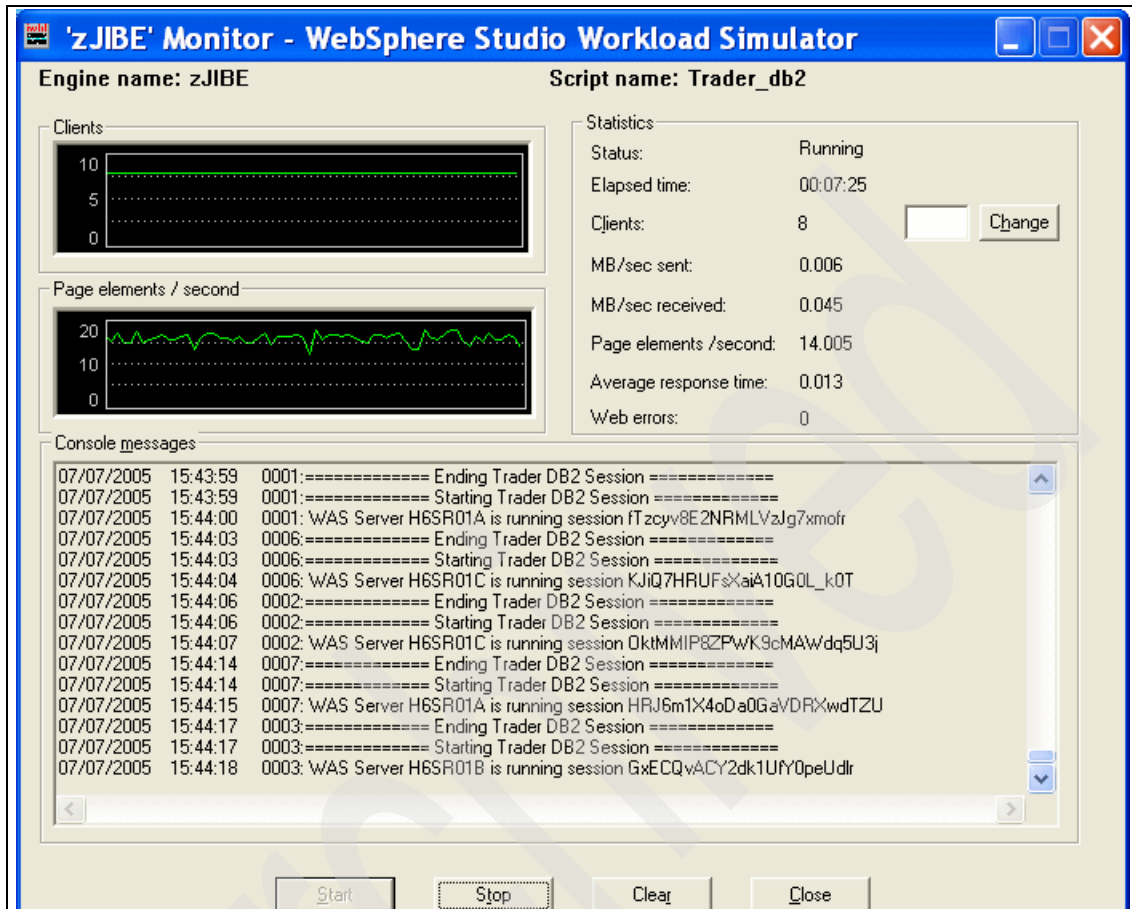


Figure 10-2 WebSphere Workload Simulator monitor screen

The monitor screen contains a section called Page elements/second. This graph shows the number of Web pages per second being returned to WebSphere Workload Simulator over time. It shows when the throughput of the test workload is interrupted due to outages in the infrastructure. We use this graph in later chapters of this book to show how the tests we ran affected the throughput of the workload.

10.2.1 The workload

We created several test scripts so that we could use a specific script to more accurately test a particular piece of our architecture.

Trader_db2

This script invokes the TraderDB2 application. It consists of:

- ▶ 50% Read Transactions
- ▶ 50% Write Transactions

This script includes the following steps:

1. Logging in to your account. This establishes a session.
2. Requesting several quotes. These requests flow on the same session established during login.
3. Making several buys and sells. These requests flow on the same session established during login.
4. Logging out. This destroys the session.

Trader_JMS

This script invokes the TraderJMS application. It consists of:

- ▶ 50% Read Transactions
- ▶ 50% Write Transactions

This script includes the following steps:

1. Logging in to your account. This establishes a session.
2. Requesting several quotes. These requests flow on the same session established during login.
3. Making several buys and sells. These requests flow on the same session established during login.
4. Logging out. This destroys the session.

So on average we were performing writes 50% of the time and reads 50% of the time. This is a higher ratio of writes than would normally be done by a customer application, but since the goal of our testing was to ensure high availability and not to try for peak performance, we felt it was more valuable to have the tests create high levels of system activity.

10.2.2 Scripts for WebSphere Studio Workload Simulator

Because we were very interested in seeing how the workload we were running would fail over sessions from one WebSphere to another, we wrote some code in the WebSphere Workload Simulator scripts to give us this information.

The scripts are included here as samples that you can use to monitor the same type of information. The scripting language in WebSphere Workload Simulator allowed us to extract the CloneID that WebSphere sets in the session cookie. The CloneID contained a WebSphere server ID that we used to show which system was receiving the current workload. We wrote scripts to run the

TraderDB2 and TraderJMS workloads, as well as scripts to extract and display the CloneIDs.

Example 10-1 Script used to run the TraderDB2 workload

```
//Trader DB2: Logon, get quotes, buy and sell, Logoff.

start_transaction("Trader DB2");

// Skip these declares unless we are running this script standalone
int html_percent = 10;
int gif_percent = 0;
.in Extract_Clone_ID_declares.jxs

console_print ("===== Starting Trader DB2 Session
=====");
// This statement clears the cookie cache so that the next time this
script is repeated
// it does not start out by sending a session affinity cookie.
clear_cookie_cache ();

// Reset these variables before starting the scenario
savedSystemID = "unset";
systemID = "unset";

startpage(1);
thinktime(0);
r = getpage("10.1.6.34", "/TraderDBWeb/Logon.jsp", 1, close, 0, start, "",
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, /**",
"Accept-Language: en-us",
"Accept-Encoding: gzip, deflate",
// "If-Modified-Since: Wed, 12 Nov 2003 11:20:54 GMT",
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
"Host: 10.1.6.34",
"Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/theme/Trader.css", 1, close, 151, start, "",
"Accept: /**",
"Referer: http://10.1.6.34/TraderDBWeb/Logon.jsp",
"Accept-Language: en-us",
"Accept-Encoding: gzip, deflate",
```



```

// "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
  "Host: 10.1.6.34",
  "Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/images/leaf1.gif", 1, close, 321, start, "",
  "Accept: */*",
  "Referer: http://10.1.6.34/TraderDBWeb/Logon.jsp",
  "Accept-Language: en-us",
  "Accept-Encoding: gzip, deflate",
// "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
  "Host: 10.1.6.34",
  "Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/images/ibm1.jpg", 1, close, 521, start, "",
  "Accept: */*",
  "Referer: http://10.1.6.34/TraderDBWeb/Logon.jsp",
  "Accept-Language: en-us",
  "Accept-Encoding: gzip, deflate",
// "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
  "Host: 10.1.6.34",
  "Connection: Keep-Alive");

endpage;
//gmc added 5/24/2005
.in Extract_Clone_ID.jxs

startpage(2);
thinktime(1000);
r =
postpage("10.1.6.34", "/TraderDBWeb/TraderSQLJDBCServlet", 1, close, 0, start, "",
  "userid=dummy&run=doPerformLogon&password=dd",
  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*",
  "Referer: http://10.1.6.34/TraderDBWeb/Logon.jsp",
  "Accept-Language: en-us",
  "Content-Type: application/x-www-form-urlencoded",

```

```

    "Accept-Encoding: gzip, deflate",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    // "Content-Length: 43",
    "Connection: Keep-Alive",
    "Cache-Control: no-cache");

get("10.1.6.34", "/TraderDBWeb/theme/Trader.css", 1, close, 7020, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/images/ibml.jpg", 1, close, 7151, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/images/leaf2.gif", 1, close, 7291, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

```

```

endpage;
.in Extract_Clone_ID.jxs
startpage(3);
thinktime(1000);
    r =
postpage("10.1.6.34","/TraderDBWeb/TraderSQLJDBCServlet",1,close,0,start,"",
    "company=Casey_Import_Export+&run=doShowQuotes",
    "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Content-Type: application/x-www-form-urlencoded",
    "Accept-Encoding: gzip, deflate",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
// "Content-Length: 45",
    "Connection: Keep-Alive",
    "Cache-Control: no-cache");

get("10.1.6.34","/TraderDBWeb/theme/Trader.css",1,close,220,start,"",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
// "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

get("10.1.6.34","/TraderDBWeb/images/leaf1.gif",1,close,321,start,"",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
// "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",

```

```

    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

    get("10.1.6.34", "/TraderDBWeb/images/ibm1.jpg", 1, close, 571, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

endpage;
.in Extract_Clone_ID.jxs

...
Lines for pages 4-20 omitted for brevity
...

startpage(21);
thinktime(1000);
    r =
postpage("10.1.6.34", "/TraderDBWeb/TraderSQLJDBCServlet", 1, close, 0, start, "",
    "run=doShowLogoff",
    "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Content-Type: application/x-www-form-urlencoded",
    "Accept-Encoding: gzip, deflate",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    // "Content-Length: 16",
    "Connection: Keep-Alive",
    "Cache-Control: no-cache");

```

```

get("10.1.6.34", "/TraderDBWeb/theme/Trader.css", 1, close, 210, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/images/leaf1.gif", 1, close, 360, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

get("10.1.6.34", "/TraderDBWeb/images/ibm1.jpg", 1, close, 480, start, "",
    "Accept: */*",
    "Referer: http://10.1.6.34/TraderDBWeb/TraderSQLJDBCServlet",
    "Accept-Language: en-us",
    "Accept-Encoding: gzip, deflate",
    // "If-Modified-Since: Fri, 07 Nov 2003 15:07:52 GMT",
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
H010818)",
    "Host: 10.1.6.34",
    "Connection: Keep-Alive");

endpage;
// Added script here 5/23/05
//.in Extract_Clone_ID.jxs

end_transaction("Trader DB2");

```

```
console_print ("===== Ending Trader DB2 Session
=====");
```

Example 10-2 shows the code that extracts the CloneID from the set-cookie header, and then displays information to the WSWs log about which clone the session is running on.

Example 10-2 Script “Extract_Clone_ID_declares”

```
// Declares for the Extract Clone ID code
// Sample cookie: "Cookie:
JSESSIONID=0001edDLRfhcRocQ6TpNkDD5ZHg:BA6F992EA41CF92100000324000000E
090C060B");
HttpResponse r;
string header, cloneID, sessionID;
string savedSystemID = "unset";
string systemID = "unset";
string systemNameA = "H6SR01A";
string systemNameB = "H6SR01B";
string systemNameC = "H6SR01C";
// The cloneIDs are long but start with these strings.
string cloneIdA = "h6sr01aSC49";
string cloneIdB = "h6sr01bSC50";
string cloneIdC = "h6sr01cSC54";
```

Example 10-3 shows the script “Extract_Clone_ID”.

Example 10-3 Script “Extract_Clone_ID”

```
// Extract and log the clone ID from the JSESSIONID cookie.
//print ("extracting Set-Cookie");
header = r.getHeader("set-cookie");
// Check to see if this request generated a set-cookie response header
//print ("print of header contents "+header);
if (StrLength(header) >= 1) {
    sessionID = StrSubString(header,StrIndexOf(header,"=")+5,StrIndexOf(header,":"));
    cloneID = StrSubString(header,StrLastIndexOf(header,":")+1,StrLastIndexOf(header, ";"));
//    print (" Clone ID = "+ cloneID + " Session ID = " + sessionID);
// Initialize the saved system ID
savedSystemID = systemID; //Save the previous system ID if one was set

// Parse the cloneID
if ( StrStartsWith(cloneID,cloneIdA) ) {
    systemID = systemNameA;
}
}
```

```

else {
    if ( StrStartsWith(cloneID,cloneIdB) ) {
        systemID = systemNameB;
    }
    else {
        if ( StrStartsWith(cloneID,cloneIdC) ) {
            systemID = systemNameC;
        }
        else {
            // Also check the systems on cluster 4
            if ( StrStartsWith(cloneID,cloneIdA4) ) {
                systemID = systemNameA4;
            }
            else {
                if ( StrStartsWith(cloneID,cloneIdB4) ) {
                    systemID = systemNameB4;
                }
                else {
                    if ( StrStartsWith(cloneID,cloneIdC4) ) {
                        systemID = systemNameC4;
                    } else {
                        systemID = "Unknown";
                    }
                }
            }
        }
    }
}

// If this session is interrupted and restarted on another server, we want to detect it.
// So compare the saved system ID to the one we just detected. If different then let us
know.
if ( StrEquals(savedSystemID,"unset") ) { } //First time through.
else {
    if ( StrEquals(systemID,savedSystemID) ) { } //Still running on same WAS server
    else {
        print ( " >>> Session " + sessionID + " was interrupted on Server " + savedSystemID +
", and restarted on server " + systemID);
    }
}
print ( " WAS Server " + systemID + " is running session " + sessionID);
}

```

When running these scripts, WebSphere Workload Simulator writes the print data continuously to a log file, as shown in Example 10-4.

Example 10-4 Sample messages written to the WSWs log file

```
06/14/05 11:21:14 - 0006: WAS Server H6SR01A is running session
PaFn65vBjIrS0d7aVSQS_lb
06/14/05 11:21:27 - 0006: WAS Server H6SR01B is running session
PaFn65vBjIrS0d7aVSQS_lb:h6sr01aSC49:h6sr01bSC50
06/14/05 11:21:27 - 0006: >>> Session
PaFn65vBjIrS0d7aVSQS_lb:h6sr01aSC49 was interrupted on Server H6SR01A,
and restarted on server H6SR01B
06/14/05 11:21:58 - 0004: WAS Server H6SR01C is running session
0001W_Jfin1kLcwP9wXL5RgpsZE
06/14/05 11:21:59 - 0006: WAS Server H6SR01C is running session
0001T3syGc4pSYRo_Euc3-OzudV
```

We created a shell script, shown in Example 10-5, in UNIX System Services that would extract data from the WebSphere Workload Simulator log file and display the percentage of the workload going to each of the three WebSphere Application Servers within our h6sr01Cluster.

Example 10-5 Sample shell script status.sh

```
#!/bin/sh
#=====
#
# This shell script summarizes the running data captured in the
# WebSphere Studio Workload Simulator log file written
# by the traderDB2 scrip
#=====
#
logfiledir="/var/iwl/log/"
temp0="/tmp/ha_replay_stats"
temp="/tmp/ha_stats"
clone="WAS Server H6SR01"
clonea="WAS Server H6SR01A"
cloneb="WAS Server H6SR01B"
clonec="WAS Server H6SR01C"
count=0
lines=10
interval=1
index=1
# Get the name of the most recent Trader log file. It's
# generated each time WSWs runs
cd $logfiledir
logfile=$(ls -t | grep zJIBE.Trader_newServlet | grep log | head -1)
grep "$clone" $logfile | grep -n "$clone" > $temp0
echo Displaying workload distribution recorded in
```


0	20	80
0	30	70
0	10	90
0	20	80
0	30	70
0	0	100
0	10	90

The count of 1 lines remaining is less than 10 - end run

10.3 Setting up TraderJMS

This section describes how to set up and install the TraderJMS application. TraderJMS uses the same application tables that TraderDB uses. Refer to the setup instructions provided in IBM Redbook *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064, for details about setting up the application tables for TraderJMS.

10.4 Configuring the required resources

In this section we describe how to configure the resources required by the TraderJMS application. This will involve the following:

- ▶ Configuring service integration bus (SIB) resources
- ▶ Configuring default messaging provider resources
- ▶ Configuring JDBC resources

10.4.1 Configuring service integration bus (SIB) resources

Begin by configuring the destinations that are needed in the bus:

1. Select **Service integration** → **Buses**.
2. Select the bus. (On our system, we used h6Bus.)
3. Select **Destinations**.
4. Click New.
 - a. Select the **Queue** option, and click Next.
 - b. Enter an **Identifier** for the destination, and click Next. (On our system we used TraderJDBCQueue.)
 - c. Use the drop-down list to assign the destination to the cluster bus member, and click Next. (The destination we used was h6sr01Cluster.)

- d. Click Finish.
5. Repeat step 4 to create a second queue destination. (On our system, we used the identifier TraderCMPQueue.)
6. Save the changes; see Figure 10-3.

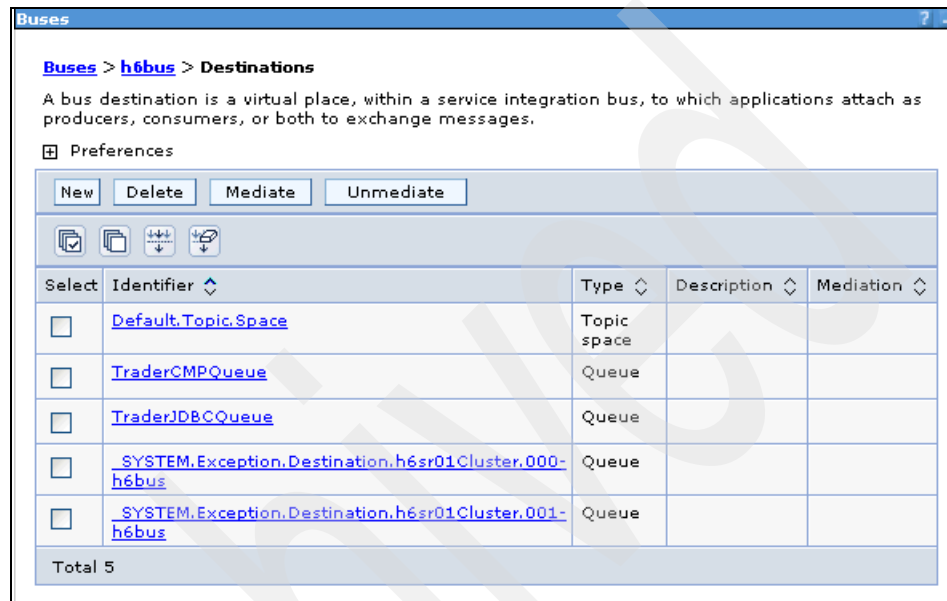


Figure 10-3 Bus destinations

10.4.2 Configure default messaging provider resources

Now you need to create the JMS resources that you require.

1. Select **Resources** → **JMS Providers** → **Default messaging**.
2. Set the scope to the cluster or cell level. (We selected the cell level on our system.)
3. Select **JMS connection factory**.
4. Click New.
 - a. For Name, we used TraderCF.
 - b. For JNDI name, we used `.jms/TraderCF`.
 - c. For Bus name, use the drop-down list to select the bus. (On our system, we selected `h6bus`.)
 - d. We also changed Persistent message reliability to Assured persistent.

- e. Then we clicked Apply.
5. To ensure that workload balancing of the JMS connections takes place, you must configure the connection pool to clean out the connections periodically. Select **Connection pool properties**.
 - a. Set Reap time to 60 seconds.
 - b. Set Unused timeout to 120 seconds.
 - c. Set Aged timeout to 120 seconds.
 - d. Click OK.
6. Return to Default messaging by selecting **Default messaging provider** from the navigation menu.

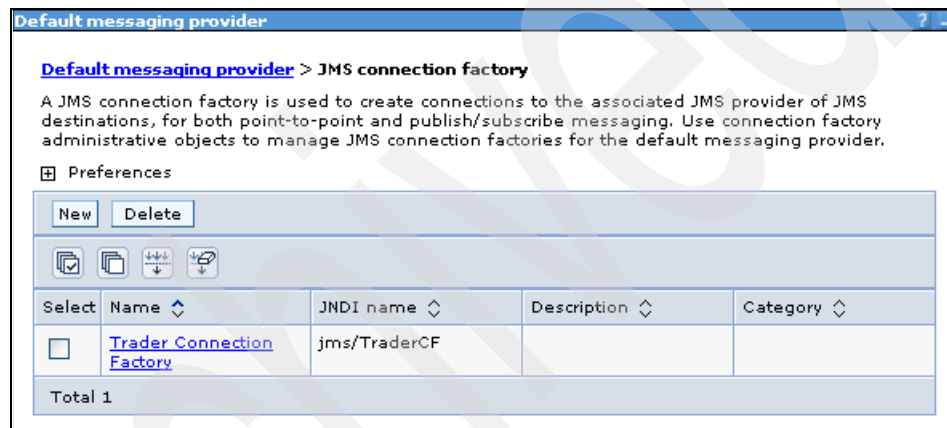


Figure 10-4 JMS connection factory

7. Select **JMS Queue**.
8. Click New.
 - a. For Name, we used TraderJDBCQueue.
 - b. For JNDI name, we used jms/TraderJDBCq.
 - c. For Bus name, we selected h6bus from the drop-down list.
 - d. For Queue name, we selected TraderJDBCQueue from the drop-down list.
 - e. Then we clicked OK.
9. Click New.
 - a. For Name, we used TraderCMPQueue.
 - b. For JNDI name, we used jms/TraderCMPq.
 - c. For Bus name, we selected h6bus from the drop-down list.

- d. For Queue name, we selected TraderCMPQueue from the drop-down list.
- e. Then we clicked OK.

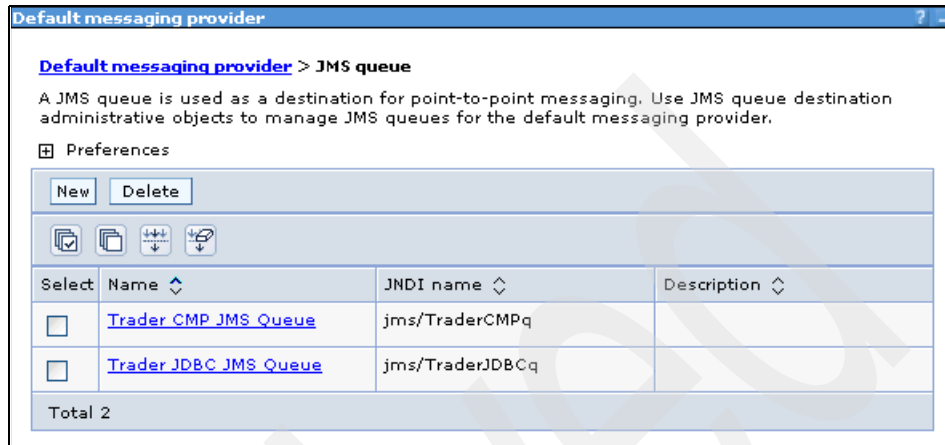


Figure 10-5 JMS queues

10. We returned to Default messaging by selecting **Default messaging provider** from the navigation menu.
11. We selected **JMS activation specification**.
12. Then we clicked New.
 - a. For Name, we used TraderJDBCActivationSpec.
 - b. For JNDI name, we used eis/TraderJDBCAs.
 - c. For Destination type, we selected Queue from the drop-down list.
 - d. For Destination JNDI name, we used jms/TraderJDBCq.
 - e. For Bus name, we selected h6bus from the drop-down list.
 - f. Then we clicked OK.
13. We clicked New.
 - a. For Name, we used TraderCMPActivationSpec.
 - b. For JNDI name, we used eis/TraderCMPAs.
 - c. For Destination type, we selected Queue from the drop-down list.
 - d. For Destination JNDI name, we used jms/TraderCMPq.
 - e. For Bus name, we selected h6bus from the drop-down list.
 - f. Then we clicked OK.

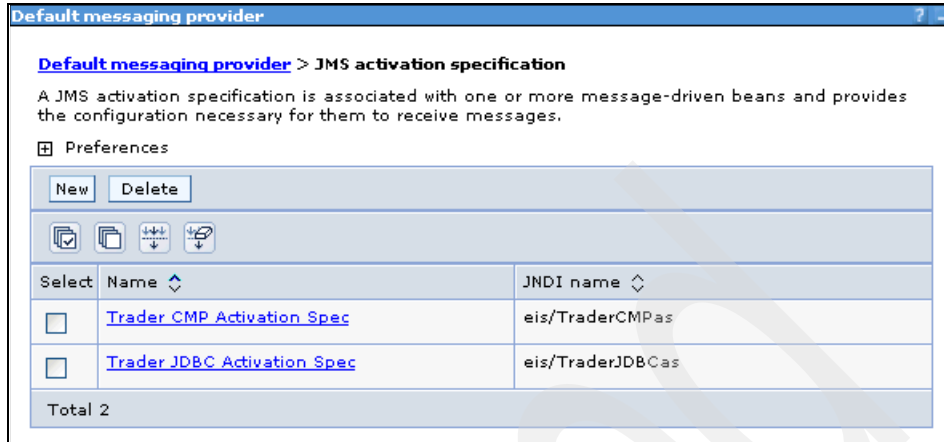


Figure 10-6 Activation specifications

14. Then we saved our changes.

10.4.3 Configure JDBC resources

The final step is to configure the JDBC resources needed by the application.

1. Select **Resources** → **JDBC Providers**.
2. Set the scope to the cluster or cell level. (We selected the cell level on our system.)
3. Click New.
 - a. For the database type, select **DB2**.
 - b. For the provider type, select **DB2 Universal JDBC Driver Provider**.
 - c. For the implementation type, select **XA data source**.
4. Click Next, and then Apply.
5. Select **Data sources**.
6. Click New.
 - a. Provide a JNDI name for the data source. (On our system we used jdbc/TraderDB2XA.)
 - b. For Database name, we used the location name for the DB2 data sharing group. On our system, this was DB8J.
 - c. For Driver type, we used type 4.
 - d. For Server name, we used HAPLEX1.ITS0.IBM.COM.
 - e. For Port number, we used 38110.

7. Click Apply.
8. Select **J2EE Connector Architecture (J2C) authentication data entries**.
9. Click New.
 - a. Select an Alias. (On our system, we entered Trader Application alias.)
 - b. Enter a User ID. Select a user id with enough authority to update the application tables.
 - c. Enter a Password.
 - d. Click OK.

Note: There is no need to specify the alias on the data source, because you will specify it later when you install the application.

10. This JDBC provider uses environment variables to locate the jar files for the driver. Ensure that the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable is set correctly.
 - a. Select **Environment** → **WebSphere Variables**.
 - b. Set the scope to the node level, as appropriate. (On our system, we had three nodes: h6nodea, h6nodeb, and h6nodec.)
 - c. Set the value for the variable DB2UNIVERSAL_JDBC_DRIVER_PATH to the path containing the JDBC driver jars.
11. Save your changes.

10.5 Install the TraderJMS application

1. Select **Applications** → **Install new application**.
2. Specify the path to the TraderJMS2005.ear file, and click Next.
3. Click Next again. You are presented with a list of thirteen steps to complete the install.
4. Go to Step 3: Select current backend ID.
 - a. Select the correct backend ID from the drop-down list.

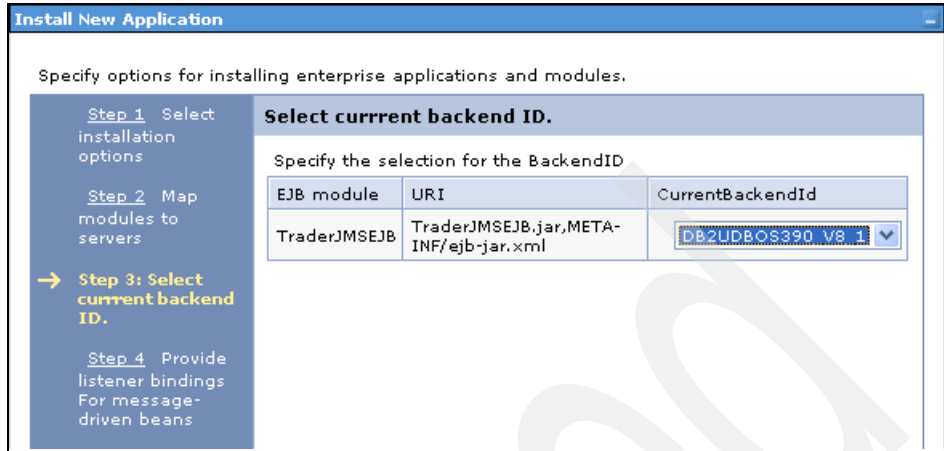


Figure 10-7 Step 3: Select current backend ID

5. Go to Step 4: Provide listener bindings for message-driven beans.
 - a. Map TraderJMS_UseJDBC to eis/TraderJDBCAs.
 - b. Map TraderJMS_UseCMP to eis/TraderCMPAs.

Select	EJB module	EJB	URI	Messaging Type	Bindings
<input type="checkbox"/>	TraderJMSEJB	TraderJMS_UseCMP	TraderJMSEJB.jar,META-INF/ejb-jar.xml	javax.jms.MessageListener	<input type="radio"/> Listener port Name <input type="text"/> <input checked="" type="radio"/> Activation Specification JNDI name <input type="text" value="eis/TraderCMPas"/> Destination JNDI Name <input type="text"/> ActivationSpec Authentication Alias <input type="text"/>
<input type="checkbox"/>	TraderJMSEJB	TraderJMS_UseJDBC	TraderJMSEJB.jar,META-INF/ejb-jar.xml	javax.jms.MessageListener	<input type="radio"/> Listener port Name <input type="text"/> <input checked="" type="radio"/> Activation Specification JNDI name <input type="text" value="eis/TraderJDBCas"/> Destination JNDI Name <input type="text"/> ActivationSpec Authentication Alias <input type="text"/>

Figure 10-8 Step 4: Provide listener bindings for message-driven beans

6. Go to Step 6: Bind message destination references to administered objects.
 - a. Map `jms/TraderJDBCQueue` to `jms/TraderJDBCq`.
 - b. Map `jms/TraderCMPQueue` to `jms/TraderCMPq`.

Select	Module	EJB	URI	Message destination object	Type	JNDI name
<input type="checkbox"/>	TraderJMSWeb		TraderJMSWeb.war,WEB-INF/web.xml	<code>jms/TraderJDBCQueue</code>	Reference	<code>jms/TraderJDBCq</code>
<input type="checkbox"/>	TraderJMSWeb		TraderJMSWeb.war,WEB-INF/web.xml	<code>jms/TraderCMPQueue</code>	Reference	<code>jms/TraderCMPq</code>

Figure 10-9 Step 6: Bind message destination references to administered objects

7. Go to Step 8: Map data sources for all 2.x CMP beans.
 - a. Map `jdbc/TraderDB2XA` to both entity beans
 - b. Set Resource authorization for both beans to container.

c. Set Authentication alias for both beans to Trader Application alias.

Select	EJB	EJB module	URI	JNDI name	Resource authorization
<input type="checkbox"/>	Company	TraderJMSEJB	TraderJMSEJB.jar,META-INF/ejb-jar.xml	<input type="text" value="jdbc/TraderDB2XA"/>	Resource authorization: Container Authentication method: DefaultPrincipalMapping h6dmnode/Trader Application alias
<input type="checkbox"/>	Customer	TraderJMSEJB	TraderJMSEJB.jar,META-INF/ejb-jar.xml	<input type="text" value="jdbc/TraderDB2XA"/>	Resource authorization: Container Authentication method: DefaultPrincipalMapping h6dmnode/Trader Application alias

Figure 10-10 Step 8: Map data sources for all 2.x CMP beans

8. Go to Step 10: Map resource references to resources.
 - a. Map both instances of `jms/TraderJMSConnectionFactory` to `jms/TraderCF`.
 - b. Map `jms/TraderJMSSClientConnectionFactory` to `jms/TraderCF`.
 - c. Map `jdbc/TraderDatasource` to `jdbc/TraderDB2XA`.
 - d. Set the Authentication alias of the data source to Trader Application alias.

Reference binding	JNDI name	Login configuration
jms/TraderConnectionFactory	jms/TraderCF	Resource authorization: Container Authentication method: none
jms/TraderConnectionFactory	jms/TraderCF	Resource authorization: Container Authentication method: none
jms/TraderJMSClientConnectionFactory	jms/TraderCF	Resource authorization: Container Authentication method: none

Figure 10-11 Step 10: Map resource references to resources

Reference binding	JNDI name	Login configuration
jdbc/TraderDatasource	jdbc/TraderDB2XA	Resource authorization: Container Authentication method: DefaultPrincipalMapping h6dmnode/Trader Application alias

Figure 10-12 Step 10 continued

9. Go to Step 13: Summary.
 - a. Click Finish.

Summary	
Summary of installation options	
Options	Values
Use Binary Configuration	No
Create MBeans for resources	Yes
Cell/Node/Server	Click here
Reload interval in seconds	
Enable class reloading	No
Application name	TraderJMS2005
Process embedded configuration	No
Validate Input off/warn/fail	warn
Directory to install application	\${APP_INSTALL_ROOT}/h6cell
Distribute application	Yes
Deploy Web services	No
Pre-compile JSP	No
Deploy enterprise beans	No


 No application modules were mapped to Web servers. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated based on the application modules which are mapped to it, therefore no Web server will route requests to this application. To change this option, select the Map modules to servers step.

Figure 10-13 Step 13: Summary

10. The application should install successfully. Select **Save to Master Configuration**.

ADMA5013: Application TraderJMS2005 installed successfully.

Application TraderJMS2005 installed successfully.

To start the application, first save changes to the master configuration.

[Save to Master Configuration](#)

Figure 10-14 Save changes



Planned infrastructure outages

This chapter describes the tests we ran to verify that the architecture we developed for high availability would work during planned outages of hardware and software components. Planned outages are used when you want to install a new software level or some other system or application maintenance.

In this chapter we show how to apply maintenance to the WebSphere Application Server for z/OS code.

In Chapter 13, “Application availability tests” on page 317, we describe how to apply maintenance to application code.

11.1 Installing new maintenance levels of WebSphere for z/OS

This section describes our method for installing new maintenance levels of WebSphere for z/OS in a sysplex.

11.1.1 Our installation's example to roll the upgrade

In our installation, to upgrade WebSphere for z/OS and roll the upgrade to all systems in the sysplex, we used the HFS structure shown in 8.4.1, “The HFS structure for upgrades” on page 232.

During WebSphere installation, we did not mount the WebSphere product HFS data set directly off the version-specific subdirectory. Instead, in the version-specific subdirectory, we created one symbolic link to the system-specific subdirectory through the use of the \$SYSNAME symbol. We followed these steps:

- ▶ We created a new WebSphere directory on the three systems:

```
MKDIR '/SC49/zWebSphere' MODE(7 5 5)  
MKDIR '/SC50/zWebSphere' MODE(7 5 5)  
MKDIR '/SC54/zWebSphere' MODE(7 5 5)
```
- ▶ We created a symbolic link:

```
BPXBATCH SH 1n -s \ $SYSNAME/zWebSphere /usr/1pp/zWebSphere/V6R0
```
- ▶ We had separate SMP/E maintenance libraries for WebSphere for z/OS, and we called them the SMP/E target libraries.
- ▶ Each system in the sysplex had its own copy of the WebSphere libraries; see Figure 11-1 on page 287. This configuration allowed us to upgrade each system without having to make updates to SYS1.PARMLIB or SYS1.PROCLIB, or to update any symbolic links

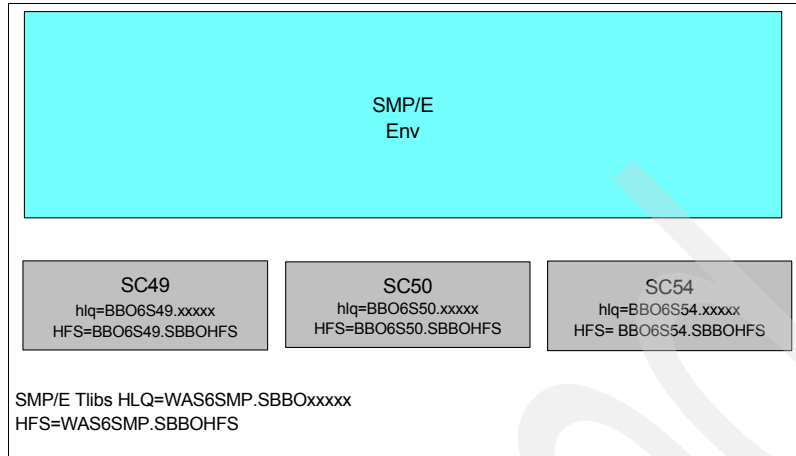


Figure 11-1 Libraries not shared

- ▶ We placed SBBOLPA and SBBoload in the dynamic LPA and added SBBOMIG and SBBOLD2 to the system LNKLIST.
- ▶ We decided to start the upgrade first on system SC54, and then on SC50. SC49 would be the last system to be upgraded, because SC48 was running the Sysplex Distributor and the Deployment Manager.
- ▶ Before shutting down the servers on the system where we were doing the upgrade, we stopped distributing to its TCP/IP stack. This was done to ensure that it did not get any new requests.
- ▶ To make sure that the servant process was ready for work, we restored the distribution back to the system TCP/IP stack after the servant process was started.
- ▶ There were no changes to the PROCS in SYS1.PROCLIB; see 8.2, “Setting up WebSphere” on page 185

11.1.2 Implementing the upgrade on system SC54

We followed these steps to implement the upgrade on system SC54:

1. We applied maintenance against our SMP/E target libraries.
2. We stopped distributing to SC54; the command must be issued in the following sequence:
 - a. From the SC49 console, we issued the following command to end distribution to SC54; see Example 11-1:

```
VARY TCPIP,TCPIPA,0,DSN=TCPIPA.SC49.TCPPARMS(VIPADL54)
```

Example 11-1 TCPIPA.SC49.TCPPARMS(VIPADL54)

```
VIPADYNAMIC
VIPADISTRIBUTE DELETE 10.1.6.150
    PORT 7888 39902 39903 39911 39912 39938 39939 80
    DESTIP 192.168.90.7
ENDVIPADYNAMIC
```

- b. From the SC54 console, we issued the following to remove the VIPA from the SC54 stack; see Example 11-2:

```
V TCPIP,TCPIPA,0,DSN=TCPIPA.SC54.TCPPARMS(VIPADL)
```

Example 11-2 TCPIPA.SC54.TCPPARMS(VIPADL)

```
VIPADYNAMIC
VIPDelete 10.1.6.150
ENDVIPADYNAMIC
```

- c. We checked the configuration.

Example 11-3 shows that SC54 is not in the configuration:

Example 11-3 SC54 IS NOT IN THE CONFIGURATION

```
D TCPIP,TCPIPA,SYSPLEX,VIPADYN
EZZ8260I SYSPLEX CS V1R6 984
VIPA DYNAMIC DISPLAY FROM TCPIPA AT SC49
IPADDR: 10.1.6.150 LINKNAME: VIPL0A010696
ORIGIN: VIPADEFINE
TCPNAME MVSNAME STATUS RANK ADDRESS MASK NETWORK PREFIX DIST
-----
TCPIPA SC49 ACTIVE 255.255.255.0 10.1.6.0 BOTH
TCPIPA SC50 BACKUP 100 DEST
TCPIPA SC54 QUIESC 255.255.255.0 0.0.0.0
```

Example 11-4 shows only the XCF address of SC49 and SC50.

Example 11-4 XCF addresses of SC49 and SC50

```
D TCPIP,TCPIPA,N,VDPT
EZZ2500I NETSTAT CS V1R6 TCPIPA 990
DYNAMIC VIPA DESTINATION PORT TABLE:
DEST IPADDR DPORT DESTXCF ADDR RDY TOTALCONN WLM FLG
10.1.6.150 00080 192.168.90.4 000 0000000000 02 B
10.1.6.150 00080 192.168.90.6 000 0000000000 01 B
10.1.6.150 07888 192.168.90.4 000 0000000000 02 B
10.1.6.150 07888 192.168.90.6 000 0000000000 01 B
10.1.6.150 38110 192.168.90.4 001 0000000030 02 B
```


10.1.6.150	38110	192.168.90.6	001	0000000005	01	B
10.1.6.150	39902	192.168.90.4	001	0000000000	02	B
10.1.6.150	39902	192.168.90.6	001	0000000000	01	B
10.1.6.150	39903	192.168.90.4	000	0000000000	02	B
10.1.6.150	39903	192.168.90.6	000	0000000000	01	B
10.1.6.150	39911	192.168.90.4	001	0000000000	02	B
10.1.6.150	39911	192.168.90.6	000	0000000000	01	B
10.1.6.150	39912	192.168.90.4	001	0000000000	02	B
10.1.6.150	39912	192.168.90.6	000	0000000000	01	B
10.1.6.150	39938	192.168.90.4	001	0000000000	02	B
10.1.6.150	39938	192.168.90.6	001	0000000000	01	B
10.1.6.150	39939	192.168.90.4	001	0000000000	02	B
10.1.6.150	39939	192.168.90.6	001	0000000000	01	B

3. We waited 10 minutes to allow any transactions that were running on SC54 to end.
4. We shut down all application servers and the node agent in SC54.

11.1.3 Rolling Upgrade process

In this section, we describe how to use the Rolling Upgrade process.

1. We backed up the WebSphere Application Server's node HFS (HFS that is built by the WebSphere dialogs and populated with new files after a PTF upgrade).

We found that if we had to back out the upgrade, it was easier and faster to restore the node's HFS from a backup rather than using the `backoutPTF.sh` shell script to back out the change.

Example 11-5 JCL to back up the node HFS

```
//SC52HABK JOB (999,P0K), 'CONWAY', CLASS=A, REGION=OM,
//          MSGCLASS=T, TIME=10, MSGLEVEL=(1,1), NOTIFY=&SYSUID
/*JOBPARM L=999, SYSAFF=SC54
//*****
/* DUMP (LOGICAL) HFS
//*****
//DUMP      PROC
//SU       EXEC  PGM=ADRDSU
//SYSPRINT DD   SYSOUT=*
//HFS1     DD    UNIT=3390, VOL=SER=T0THF6, DISP=SHR
//          PEND
//*
//STEP1    EXEC  DUMP
//SU.HFSOUT DD   DSN=OMVS.SC54.WASV6.HABSOC.HFS.SEQ.D040228,
```

```
//          DISP=(NEW,CATLG,DELETE),SPACE=(CYL,(350,50),RLSE),
//          UNIT=3390,VOL=SER=TOTHF6
//SU.SYSIN  DD *
DUMP DATASET (INCLUDE(OMVS.SC54.WASV6.HABSOC.HFS)) -
        COMPRESS TOL(ENQF) -
        LOGINDDNAME(HFS1) OUTDDNAME(HFSOUT) ALldata(*) ALLEXCP
/*
```

2. In our configuration we ran with data sets SBBOMIG and SBBOLD2 in the system LNKLIST. Because these libraries are allocated by LLA, they cannot be renamed without stopping the LLA address space.

For this reason, we made backup copies of these data sets and copied in the updates directly from the SMP/E target libraries; see Example 11-6.

Example 11-6 JCL to back up and refresh data sets on LNKLIST

```
//SC52IEBC JOB (999,P0K),'IEB COPY',NOTIFY=&SYSUID,
//          CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*
//*MAKE BACKUP COPIES OF THE WEBSHERE LNKLISTED DATASETS
//*
//CPYBACK EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSUT3 DD UNIT=VIO,SPACE=(CYL,(25,10))
//SYSUT4 DD UNIT=VIO,SPACE=(CYL,(25,10))
//IN01 DD DISP=SHR,DSN=BB06S54.SBBOLD2
//IN02 DD DISP=SHR,DSN=BB06S54.SBBOMIG
//OUT01 DD DSN=BB06S54.SBBOLD2.OLD,DISP=(,CATLG),
//        LIKE=BB06S54.SBBOLD2,
//        UNIT=3390,VOL=SER=WS5S52
//OUT02 DD DSN=BB06S54.SBBOMIG.OLD,DISP=(,CATLG),
//        LIKE=BB06S54.SBBOMIG,
//        UNIT=3390,VOL=SER=WS5S52
//SYSIN DD *
COPY INDD=IN01,OUTDD=OUT01
COPY INDD=IN02,OUTDD=OUT02
/*
//*COPY IN THE LATEST MODULES FOR ALL LNKLISTED DATASETS
//*
//COPYNEW EXEC PGM=IEBCOPY,REGION=0M
//SYSPRINT DD SYSOUT=*
//IN03 DD DSN=WAS6SMP.SBBOLD2,DISP=SHR
//IN04 DD DSN=WAS6SMP.SBBOMIG,DISP=SHR
//OUT03 DD DSN=BB06S54.SBBOLD2,DISP=SHR
```

```
//OUT04 DD DSN=BB06S54.SBBOMIG,DISP=SHR
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(25,10)),DISP=(NEW,DELETE)
//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(25,10)),DISP=(NEW,DELETE)
//SYSIN DD *
COPY INDD=((IN03,R)),OUTDD=OUT03
COPY INDD=((IN04,R)),OUTDD=OUT04
/*
```

3. We unmounted the WebSphere product HFS data set (BBO6S54.SBBOHFS was renamed to BBO6S54.SBBOHFS.OLD) .
4. We renamed the rest of the existing WebSphere libraries to a different name in case we had to back out the upgrade (BBO6S54.xxxxxx was renamed to BBO6S54.xxxxx.OLD).
5. We copied the rest of the SMP/E target libraries and renamed to the original names (WAS6.SMP.SBBOxxxxx was copied with new name BBO6S54.SBBOxxxx); see Example 11-7.

Example 11-7 JCL to copy updates from SMP target libraries

```
//COPYBB50 JOB (999,POK),'COPY',CLASS=A,REGION=0K,
//          MSGCLASS=T,TIME=10,MSGLEVEL=(1,1),NOTIFY=&SYSUID
/*JOBPARM L=999,SYSAFF=SC69
/*
/* THIS JOB COPIES THE WAS V5 TARGET LIBRARIES
/*
//COPY EXEC PGM=ADDRSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
//FROMDAS1 DD UNIT=3390,DISP=SHR,VOL=SER=WA5T10
//FROMDAS2 DD UNIT=3390,DISP=SHR,VOL=SER=WA5T11
//TODASD DD UNIT=3390,DISP=SHR,VOL=SER=WS5S50
//SYSIN DD *
COPY DATASET(INCLUDE(WAS6SMP.SBBO* -
EXCLUDE(WAS6SMP.SBBOLD2, -
WAS6SMP.SBBOMIG -
)) -
INDD(FROMDAS1,FROMDAS2) OUTDD(TODASD) -
TOLERATE(ENQF) -
RENAMEU(WAS6SMP.**,BBO6S54.**)) -
CATALOG -
SHARE ALLEXCP ALLDATA(*) CANCELERROR
/*
//
```

6. We mounted the new WebSphere HFS data set (BBO6S54.SBBOHFS) at /usr/lpp/zWebSphere. We entered the following command from TSO:

```

MOUNT FILESYSTEM('BB06S54.SBB0HFS') TYPE(HFS)
MODE(RDWR) MOUNTPOINT('/SC54/zWebSphere') UNMOUNT

```

7. Since we ran with SBBOLPA and SBBLOAD in the dynamic LPA, we had to delete all members (one by one), and then add the new updated members from the SMP/E target libraries.

Example 11-8 shows the JCL to execute a REXX exec called MEMBERL to build the LPA DELETE cards that can be used to delete all the WebSphere modules from the dynamic LPA.

Example 11-8 JCL to build a list of dynamic LPA modules

```

//SC52MEML JOB (999,POK),'CONWAY',CLASS=A,MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID,REGION=OM,
//          MSGCLASS=T
/*JOBPARM SYSAFF=SC52
//STEP1 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSEXEC DD DISP=SHR,DSN=SYS1.OS390.CLIST
//SYSTSPRT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
MEMBERL BB06S54.SBBLOAD.OLD SYS1.PARMLIB PROG#1
MEMBERL BB06S54.SBBOLPA.OLD SYS1.PARMLIB PROG#2
/*
//

```

Example 11-9 shows the REXX exec MEMBERL.

Example 11-9 REXX exec to format a list of LPA modules

```

/* REXX EXEC */
/* Create a list of members of a data set. */
/* This exec used to build commands to remove WASV6 */
/* dynamic LPA modules */
/* */
/* See HELP (below) for syntax of this command. */
/* */
trace o
ARG pdsin pdsout memname /* Supply input and outp.*/
if pdsin = '' | pdsout = '' | memname = '', /* Need help? */
| pdsin = 'HELP' | pdsin = '?' then call HELP
call CHECK_ARGS /* Check datasets. */
x = OUTTRAP(member.) /* "Trap" output. */
"LISTD ('"pdsin"') MEMBERS" /* Get list of members. */

```

```

x = OUTTRAP(OFF)                /* Stop "trapping" output*/
i = 1                          /* Initialize LISTD cnt. */
do while i <= member.0        /* Step thru LISTD list. */
  if SUBSTR(member.i,3,6) = MEMBER then do /* Find word 'MEMBER'. */
    z = i+1                    /* Line after 'MEMBER'. */
    i = member.0 + 1          /* Quit DO WHILE. */
    outcnt = 1                /* Set output counter. */
    do while z <= member.0    /* Step thru members. */
      memb = SUBSTR(member.z,3,LENGTH(member.z)-2)
      call MAKE_OUTPUT        /* Stack the results. */
      z = z + 1              /* Increment member cnt. */
    end                      /* END for DO WHILE Z.. */
  end                          /* END for THEN DO. */
  else i = i+1                /* Increment LISTD cnt. */
end                             /* END for DO WHILE I.. */
/* Use TSO to allocate */
/* and write the member. */

ADDRESS TSO "ALLOC FI(OUTDS) DA('pdsout'('memname')) SHR REU"
"EXECIO * DISKW OUTDS (STEM newvar. FINIS)"
exit                             /* Done!
*/

HELP:                            /* Give help.
*/
say '*****'
say
say 'MEMBERL will make a list of member names of a partitioned
dataset.'
say
say 'The correct syntax of MEMBERL is:'
say
say 'MEMBERL <PDSin> <PDSout> <member>'
say
say 'where:'
say '   PDSin = name of dataset whose members will be listed.'
say '   PDSout = name of dataset that will hold the output.'
say '   member = name of the member of PDSout that will'
say '             contain the list of members of PDSin'
say
say '   Note: If all 3 are not supplied, HELP will be supplied.'
say
say 'For example, to list the members of the dataset CCSG.VTAMLST,'
say 'and place the output list in the member MEMLIST of the dataset'
say 'MYID.PDS1, use the command:'
say

```

```

say '          MEMBERL CCSG.VTAMLST MYID.PDS1 MEMLIST      '
say '*****'
exit
return

CHECK_ARGS:                                /* Verify before starting*/
x = LISTDSI("pdsin")                        /* Check input dataset. */
if SYSREASON ^=0 then do                    /* LISTDSI not good. */
  say 'Your input dataset, 'pdsin', is not valid.' /* Explain. */
  say 'REASON CODE return from LISTDSI function is:' SYSREASON
  say 'Please verify and re-run MEMBERL.'
  exit
end
else do
  x = LISTDSI("pdsout")                      /* Check output dataset. */
  /*
  if SYSREASON ^=0 then do                    /* LISTDSI not good. */
  /*
    say 'Your output dataset, 'pdsout', is not valid.' /* Explain. */
  /*
    say 'REASON CODE return from LISTDSI function is:' SYSREASON
    say 'Please verify and re-run MEMBERL.'
    exit
  end
end
return;

MAKE_OUTPUT:
PARSE VAR memb part1 part2
if part2 = '' then
do
cmd1='LPA DELETE MOD('part1') FORCE=YES'
end
else
do
PARSE VAR part2 p1 '(' part2a
cmd1='LPA DELETE MOD('part1') FORCE=YES'
cmd2='LPA DELETE MOD('part2a' FORCE=YES'
newvar.outcnt = cmd2                        /* Make stack of cmds*/
outcnt = outcnt + 1                          /* Increment output count*/
end
newvar.outcnt = cmd1                          /* Make stack of cmds*/
outcnt = outcnt + 1                          /* Increment output count*/
return

```

Two new members (PROG#1 and PROG#2) will be created in SYS1.PARMLIB. They contain statements to delete the WebSphere modules in the dynamic LPA.

```
LPA DELETE MOD(BBOBOA) FORCE=YES
LPA DELETE MOD(BBOCLSPC) FORCE=YES
LPA DELETE MOD(BBOCOMM) FORCE=YES
LPA DELETE MOD(BBOCORBA) FORCE=YES
LPA DELETE MOD(BBOCTL) FORCE=YES
...
```

From the SC54 console, we issued the following MVS commands to delete the LPA modules:

```
T PROG=#1
T PROG=#2
```

8. We built another PROGXX member in SYS1.PARMLIB (prog#3, for this example), and added the following statements:

```
LPA ADD MASK=* DSNAME(BB06S54.SBB0LPA)
LPA ADD MASK=* DSNAME(BB06S54.SBB0LOAD)
```

We issued a T PROG=#3 to load the new updated WebSphere modules into the dynamic LPA.

9. Then we refreshed library lookaside (LLA).
10. We restarted WebSphere for z/OS and the application servers on SC54.
11. We waited for the servant process to come up.

```
Server SERVANT PROCESS h6sr01c open for e-business
```

12. We restored the IP distribution to SC54. The commands have to be issued in the following sequence:

- a. From SC54, we issued the following command; see Example 11-10:

```
v tcpip,tcpipa,o,dsn=TCPIPA.SC54.TCPPARMS(VIPADEF)
```

Example 11-10 TCPIPA.SC54.TCPPARMS(VIPADEF)

```
VIPADYNAMIC
VIPABACKUP 80 10.1.6.150
ENDVIPADYNAMIC
```

- b. see Example 11-11:

```
v tcpip,tcpipa,o,dsn=TCPIPA.SC49.TCPPARMS(VIPADF54)
```

Example 11-11 TCPIPA.SC49.TCPPARMS(VIPADF54)

```
VIPADYNAMIC
VIPADISTRIBUTE DEFINE 10.1.6.150
    PORT 7888 39902 39903 39911 39912 39938 39939 80
    DESTIP 192.168.90.7
ENDVIPADYNAMIC
```

13. We repeated this process for SC50.

14. We repeated the same process for SC49, but with the following changes:

- a. SC49 is running the deployment manager. In addition to backing up the node HFS data set before the upgrade, you should back up the deployment manager HFS data set as well.
- a. To stop distributing to SC49, we issued the following command; see Example 11-12 on page 296:

```
v tcpip,tcpipa,o,dsn=TCPIPA.SC49.TCPPARMS(VIPADL49)
```

Example 11-12 TCPIPA.SC49.TCPPARMS(VIPADL49)

```
VIPADYNAMIC
VIPADISTRIBUTE DELETE 10.1.6.150
    PORT 7888 39902 39903 80 39911 39912 39938 39939
    DESTIP 192.168.90.4
ENDVIPADYNAMIC
```

- b. To start distributing to SC49, we issued the following command;

```
v tcpip,tcpipa,o,dsn=TCPIPA.SC49.TCPPARMS(VIPADF49)
```

Example 11-13 TCPIPA.SC49.TCPPARMS(VIPADF49)

```
VIPADYNAMIC
VIPADISTRIBUTE DEFINE 10.1.6.150
    PORT 7888 39902 39903 80 39911 39912 39938 39939
    DESTIP 192.168.90.4
ENDVIPADYNAMIC
```

Unplanned outages

We ran a series of tests to verify that the architecture we developed for high availability would continue to provide unbroken application service during a wide range of unplanned outages. This chapter presents a summary of the tests we ran, then goes on to describe each test in more detail.

In all cases, the application remains continuously available from the user's perspective, even though a component may take an unplanned outage.

For most unplanned outages, the system recovers automatically—but for some, the system does not recover without intervention. For those cases, we discuss the steps you should take to recover the failed component.

12.1 Summary of unplanned outage tests

Every time we ran a test, we first started a workload using the WebSphere Workload Simulator (see 10.2, “WebSphere Workload Simulator” on page 260).

With the workload running at full volume, we would then start the test by causing an outage in a particular component in our system. Table 12-1 on page 298 summarizes the tests we ran.

We did not test all the situations that we tested in the first version of this redbook. For this update we felt that we did not need to test the hardware failures or some of the software failures that had nothing to do with WebSphere, since those components had not changed since the last version. We also excluded failures generated from using the firewall.

Table 12-1 Our unplanned outage tests

Test	Description	How we killed it	Results
Edge Server	Edge Server failure	Logoff VM guest	A failure of the primary causes the backup to take over the workload. Application users do not notice this failure.
Proxy Server	Proxy Server failure	Logoff VM guest	A failure in one causes the other to take over the workload. Application users do not notice this failure.
Single WebSphere	Simulate failure of WebSphere server	Cancel the daemon, control, and servant process	Sessions are resumed on another application server. New requests in that session will be routed to other application servers in the sysplex.
WebSphere Servant process	Simulate failure of WebSphere servant process	Cancel a servant process	Sessions are resumed on the other servant process. New requests will be routed to the other servant process or other application servers in the sysplex.
Single WebSphere with Peer Recovery	Simulate failure of WebSphere server	Cancel the daemon, control, and servant process	Current application requests that were running on this server will time out. New requests will be routed to other application servers in the sysplex until this server is restarted in another LPAR.
Two WebSpheres	Simulate failure of two WebSphere servers	Cancel the daemon, control, and servant process	Current application requests that were running on those servers will time out. New requests will be routed to the remaining server in the sysplex.

Test	Description	How we killed it	Results
Sysplex Distributor	Simulate TCP/IP stack failure or LPAR failure	Cancel TCP/IP	VIPA is automatically switched to another LPAR, where the backup Sysplex Distributor takes over.
Deployment Manager with Peer Recovery	Simulate failure of WebSphere DM	Cancel the daemon, control, and servant process	WebSphere application servers continue to function normally, with no interruption to users. WebSphere administration will stop until DM restarts in another LPAR.

12.2 Edge server

We ran the Network Dispatcher component of the WebSphere Edge Components in a Linux guest. Another Linux guest acted as a hot backup running the same software. If the Network Dispatcher died, the backup would take over for it and no work would be lost.

Our test

1. We started our test workload and allowed it to reach full capacity.
2. The Edge server running on Linux guest LNXES1 was our primary Network Dispatcher. We killed this server by logging off the VM system it was running on.
3. On the WebSphere Studio Workload Simulator screen, as shown in Figure 12-1, we observed that the throughput of our test workload fell to zero very briefly, and then recovered back to original throughput levels as LNXES2 became the active Edge server. No transactions timed out during this recovery.

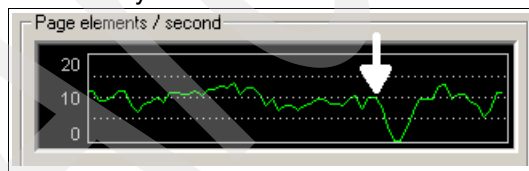


Figure 12-1 WSWs shows drop in throughput and then recovery

4. We restarted the failed Edge server by rebooting LNXES1.
5. When Linux finished booting, the Edge server started and once more became the primary Network Dispatcher.

As shown in Figure 12-2, we observed that the throughput of our test workload fell to zero very briefly, and then recovered back to original

throughput levels as LNXES1 became the active Edge server. No transactions timed out during this recovery.

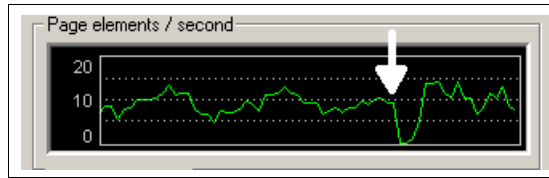


Figure 12-2 WSWS shows drop in throughput as LNXES1 resumes its role as primary Edge server

12.3 Proxy server

We ran the Proxy server component of the WebSphere Edge Components in a Linux guest. Another Linux image was also running the same software. There is no automatic failover between Proxy servers, but since there were two of them sharing the workload, there was no single point of failure.

Our test

1. We started our test workload and allowed it to reach full capacity.
2. The Proxy server running on Linux guest LNXPR1 was our primary Proxy server. We killed this Proxy server by logging off the VM system it was booted under.
3. On the WebSphere Studio Workload Simulator screen, as shown in Figure 12-3, we observed that there was a small interruption in throughput of our test workload as the Edge Server realized that proxy LNXPR1 was no longer active. Five requests timed out that had previously been sent to LNXPR1 before the Edge Server realized it was no longer active.

All the requests sent by the Proxy server were now routed to the second Proxy server running on LNXPR2, and transaction throughput resumed its normal levels.

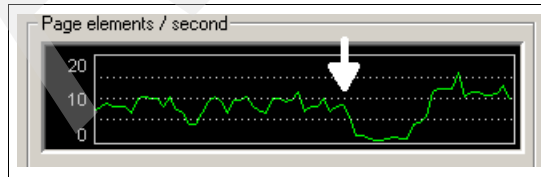


Figure 12-3 WSWS shows drop in throughput, and then recovery

4. We restarted the failed Proxy server by rebooting LNXPR1.

5. When Linux finished booting, the Proxy server started and was once more recognized as active by the Edge Server. Then the Edge Server resumed spraying work between both Proxy servers.

12.4 HTTP server

We ran the IBM HTTP server as a Linux guest. Another Linux guest also ran the HTTP server in another LPAR. Because the HTTP Servers were behind the proxies, and the Network Dispatcher sprayed requests to the two proxies, each HTTP server handled half the requests coming from our Edge server.

In this setup, the HTTP servers became single points of failure. Therefore, we did not test what happens when one was killed, because we knew that all throughput through that proxy-HTTP pair would time out, causing 50% of the total throughput to time out.

12.5 Single WebSphere

Disabling a control process will cause immediate termination of all applications running in the servant process under that control process. User transactions already queued to the servant process of that control process will time out.

All application workload is transferred to the remaining two LPARs where the application servers remain active. Users who had sessions in progress with the failed application server will continue their sessions.

Our test

1. We started our test workload and allowed it to reach full capacity.
2. WebSphere was running on all three LPARs, sharing the application load equally. We killed the WebSphere H6SR01B running in LPAR SC50.
3. On the WebSphere Studio Workload Simulator screen, as shown in Figure 12-4 on page 302, we observed that there was a very small interruption in throughput of our test workload. This interruption was due to the time it took the WebSphere plug-in to realize that H6SR01B was down, and also due to the time it took to recover the inflight sessions on another application server. No transactions timed out during this recovery.

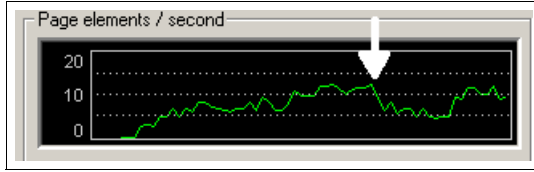


Figure 12-4 WWSWS shows a slight drop in throughput, and then recovery

4. The WebSphere plug-in running in the two HTTP servers realized that WebSphere H6SR01B was down, and it prevented workload from being sent to it. All workload was now being sent to the other two WebSpheres.

As shown in Figure 12-5, all three servers processed requests until we brought down server H6SR01B; then the requests were routed to the other servers.

H6SR01A %	H6SR01B %	H6SR01C %
60	10	30
70	0	30
60	0	40
70	0	30
40	0	60
40	0	60
30	0	70
40	0	60
80	0	20
90	0	10
50	0	50

Figure 12-5 H6SR01B down

5. WebSphere resumed the sessions that were in progress with H6SR01B on another application server.
We restarted WebSphere H6SR01B. After it became active, the workload started flowing to it, as well.

12.6 WebSphere servant process

In previous Versions of WebSphere Application Server, disabling a servant process caused immediate termination of all application requests running in that servant process. In Version 6 of WebSphere Application Server, the Data Replication Service (DRS) recovers the persistent services for the inflight

transactions. The failed inflight transactions are re-routed to the other active servant regions.

1. We started our test workload on WSWs and allowed it to reach full capacity.
2. WebSphere was running on all three LPARs, sharing the application load equally, as shown in Figure 12-6. We killed one of the three WebSphere servant regions, H6SR01BS, running in LPAR SC50.

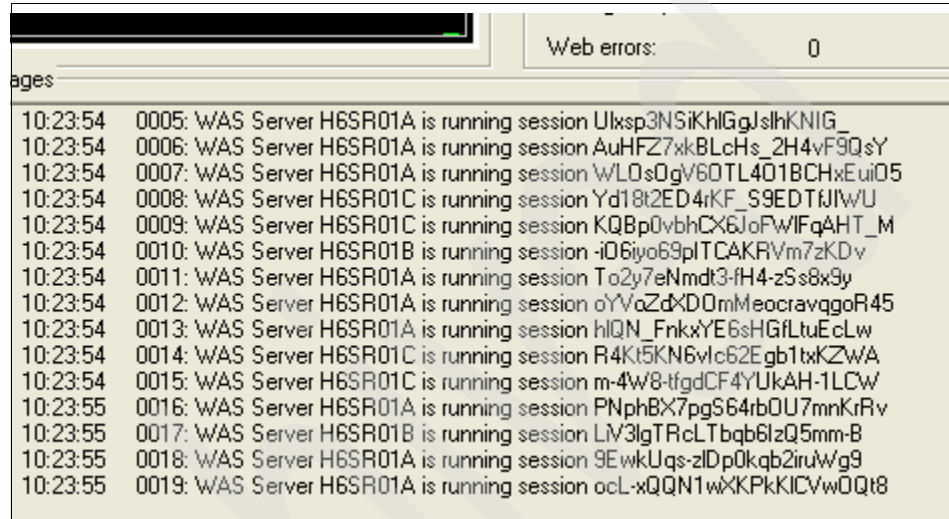


Figure 12-6 WSWs screen shot of active sessions

3. On the WebSphere Studio Workload Simulator screen, as shown in Figure 12-7, we observed that there was a very small interruption in throughput of our test workload.

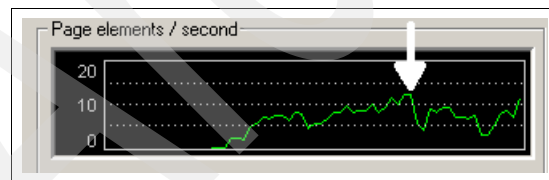


Figure 12-7 WSWs shows slight drop in throughput, and then recovery

4. This small interruption was due to the amount of time it took WebSphere to realize that a servant process was gone and to restart its work on the remaining servant process. No transactions timed out during this recovery.
5. WLM automatically started a new servant process to replace the one we killed on LPAR SC50.

12.7 Two WebSpheres

Disabling the control processes of two WebSphere application servers, in two different LPARs, will cause immediate termination of all applications running in the servant process under those control processes. All user transactions already queued to the servant process of those control processes will time out.

All application workload is transferred to the remaining LPAR where the WebSphere application server remains active. Users who had sessions in progress with the failed application servers will resume their sessions on the remaining application server.

Our test

1. We started our workload and allowed it to reach full capacity.
2. WebSphere was running on all three LPARs, sharing the application load equally. We killed the WebSphere H6SR01A running in LPAR SC49 and the WebSphere H6SR01C running in LPAR SC54.
3. On the WebSphere Studio Workload Simulator screen, as shown in Figure 12-8, we observed that there was a degradation in throughput of our test workload for several seconds, then a recovery. This degradation resulted from the following conditions:
 - Not getting a response from the requests already running on WebSpheres A and C
 - The time it took the WebSphere plug-in to realize that A and C were down
 - The time it took to recover the inflight sessions on WebSphere B

Five transactions timed out during this recovery.

Note that throughput did not fully recover to the same levels as before the failure. This was because application server B could not process the transactions fast enough by itself, since the other two (of the three) WebSphere application servers were down.

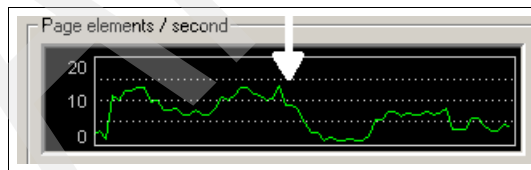


Figure 12-8 WWSWS shows drop in throughput, then limited recovery

4. WebSphere resumed the sessions that were in progress with A and C on application server B. In our test, five sessions total (three sessions from A and two from C) were resumed on B.

Example 12-1 shows lines from the WebSphere stdout for H6SR01A, generated by the DRS and HTTP Session Mgr traces. We show the trace entries for only one of the sessions.

Example 12-1 Creating the original session in H6SR01A - Generate the session ID and set the cookie

```
Trace: 2004/02/05 14:15:42.034 01 t=6BD088 c=194.1 key=P8 (13007002)
  FunctionName: com.ibm.ws.webcontainer.httpsession.SessionContext
  SourceId: com.ibm.ws.webcontainer.httpsession.SessionContext
  Category: DEBUG
  ExtendedMessage: SessionContext:createSession: generated new
sessionid = 5z0Z4pa4uMs0Guka8s8brz5
```

```
Trace: 2004/02/05 14:15:42.113 01 t=6BD088 c=194.1 key=P8 (13007002)
  FunctionName: com.ibm.ws.webcontainer.httpsession.SessionContext
  SourceId: com.ibm.ws.webcontainer.httpsession.SessionContext
  Category: DEBUG
  ExtendedMessage: SessionContext.setCookie
:00015z0Z4pa4uMs0Guka8s8brz5:BA6F992EA41CF921000003240000000E090C060B
```

Example 12-2 and Example 12-3 on page 306 show lines from the WebSphere stdout for H6SR01B, where the sessions are being resumed after the failure of application servers A and C.

We show the trace entries for only one of the sessions.

Example 12-2 Resuming the session in H6SR01B - Find a valid cookie, but cannot find the session in the cache for H6SR01A

```
Trace: 2004/02/05 14:17:15.467 01 t=6C9E88 c=1E9.1 key=P8 (13007002)
  FunctionName: com.ibm.ws.webcontainer.httpsession.SessionContext
  SourceId: com.ibm.ws.webcontainer.httpsession.SessionContext
  Category: DEBUG
  ExtendedMessage: SessionContext.getRequestedSessionId: Cookie Id =
00015z0Z4pa4uMs0Guka8s8brz5:BA6F992EA41CF921000003240000000E090C060B
```

```
Trace: 2004/02/05 14:17:15.479 01 t=6C9E88 c=1E9.1 key=P8 (13007002)
  FunctionName: com.ibm.ws.webcontainer.httpsession.SessionContext
  SourceId: com.ibm.ws.webcontainer.httpsession.SessionContext
  Category: DEBUG
  ExtendedMessage: BackedHashtable. session not found in cache
5z0Z4pa4uMs0Guka8s8brz5
```

Example 12-3 Resuming the session in H6SR01B - Read session object from DRS Cache into new servant process; send new cookie with both old and new cloneIDs

```
Trace: 2004/02/05 14:17:15.493 01 t=6C9E88 c=1E9.1 key=P8 (13007002)
  FunctionName: com.ibm.ws.webcontainer.httpsession.DRSHttpSessCache
  SourceId: com.ibm.ws.webcontainer.httpsession.DRSHttpSessCache
  Category: ENTRY
  ExtendedMessage: getEntry
5z0Z4pa4uMs0Guka8s8brz5default_host/TraderDBWeb
```

```
Trace: 2004/02/05 14:17:15.971 01 t=6C9E88 c=1E9.1 key=P8 (13007002)
  FunctionName: com.ibm.ws.webcontainer.httpsession.SessionContext
  SourceId: com.ibm.ws.webcontainer.httpsession.SessionContext
  Category: DEBUG
  ExtendedMessage: SessionContext.setCookie
:00025z0Z4pa4uMs0Guka8s8brz5:BA6F992EA41CF921000003240000000E090C060B:B
A721E9FBF5DB2A3000001C000000001090C060B
```

Example 12-4 shows lines from the WebSphere Studio Workload Simulator log, where the test scripts detect that a session has been resumed on a new server.

Example 12-4 WSWS log entries showing session being resumed

```
02/05/04 09:17:16 - 0003: WAS Server H6SR01B is running session
00025z0Z4pa4uMs0Guka8s8brz5:BA6F992EA41CF921000003240000000E090C060B
02/05/04 09:17:16 - 0003: >>> Session
00025z0Z4pa4uMs0Guka8s8brz5:BA6F992EA41CF921000003240000000E090C060B
was interrupted on Server H6SR01A, and restarted on server H6SR01B
```

5. We restarted WebSpheres A and C and after they became active, the workload started flowing to them, as well.

12.8 Sysplex Distributor

A TCP/IP stack failure will cause the Sysplex Distributor (a TCP/IP component) to also fail. We failed the primary Sysplex Distributor on LPAR SC48, and that caused a backup to take over on LPAR SC50. The application work continued to be routed to the two remaining LPARs. All user transactions already running on WebSphere H6SR01A in the failed LPAR will time out. New application sessions were created on the two remaining LPARs. Users who had sessions in progress with the WebSphere on LPAR SC48 server will resume their sessions on one of the two remaining WebSpheres.

After TCP/IP is restarted on its LPAR, then normal workflow will return to the application server on this LPAR.

Our test

1. We started our test workload and allowed it to reach full capacity.
2. WebSphere was running on all three LPARs, sharing the application load equally. We killed TCP/IP on LPAR SC48.
3. On the WebSphere Studio Workload Simulator screen, as shown in Figure 12-9, we observed that there was a small reduction in throughput of our test workload.

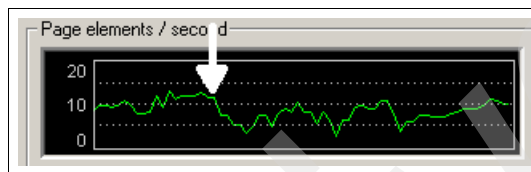


Figure 12-9 WWSWS shows a drop in throughput, then recovery

4. However, we noted some strange results. We expected that the sessions in progress on H6SR01A would be resumed on one of the other two WebSpheres, as in the other tests above, but that did not happen. Instead, transactions that were in session with WebSphere H6SR01A on LPAR SC48 timed out, and continued to time out as each new transaction was run.

Even though we had DRS replication set to “immediate”, we expected that some session data would not be replicated to the other application servers once TCP/IP had been stopped. But we expected that those sessions on H6SR01A would be resumed by the other WebSpheres. DRS traces showed that the sessions were being resumed by the other WebSpheres, but the transactions were still timing out, which implied that WebSphere was not returning data for those transactions.

We found DRS was actually failing over the sessions, but taking 3 minutes and 30 seconds to do it. After failing over one request in the session, DRS took another 3:30 to fail over the next request in the session. In our tests the requests were timing out since our time-out value in WWSWS was set to 60 seconds.

12.9 Deployment Manager with Peer Recovery

The Deployment Manager application server is not part of the WebSphere application runtime, so failures there had no effect on our test workload.

However, we wanted to test that a failure in Deployment Manager could also be recovered in a highly-available mode.

For this reason, we enabled the Peer Recovery feature of WebSphere so that the failed Deployment Manager application server would automatically restart on another LPAR.

Our test

1. Since the Deployment Manager had no effect on workload running in other WebSphere application servers, we did not run a test workload during this test.
2. Our DM was running on node SC48. We stopped the WebSphere Application Server that ran the Deployment Manager application. The Administrative Console (GUI) will not connect to WebSphere until the new application server for DM starts in another LPAR. There is no session backup for the Deployment Manager, so sessions will be lost when it fails.
3. We started DM on node SC50 (we could also have started it on SC52).

Example 12-5 SDSF display of Deployment Manager controller and servant running in SC52 after restart

SDSF DA	SC52	SC52	PAG	0	SIO	6	CPU	8/	8	LINE	1-2	(2)
COMMAND	INPUT	===>										SCROLL =
NP	JOBNAME	StepName	ProcStep	JobID	Owner	C	Pos	DP	Real	Paging		
	WHDMGR	WHDMGR	BBOCTL	STC09231	WHACRU		NS	FE	27T	0.00		
	WHDMGRS	WHDMGRS	BBOSR	STC09232	WHASRU		IN	FE	80T	0.00		

4. We started the WebSphere Administrative Console to verify that the DM was working.
5. We did find that the server status in the Admin Console sometimes showed inconsistent information. It showed the Node Agents as started, but did not show the application servers as started (when in fact, they were); see Figure 12-10 on page 309.

Application Servers

An application server is a server which provides services required to run enterprise applications. ⓘ

Total: 8

⊞ Filter

⊞ Preferences

<input type="checkbox"/>	Name ▾	Node ▾	Status ▾ ↕
<input type="checkbox"/>	whsr01a	whnodea	⊘
<input type="checkbox"/>	whsr01b	whnodeb	✖
<input type="checkbox"/>	whsr01c	whnodec	➔
<input type="checkbox"/>	whsr02b	whnodeb	✖
<input type="checkbox"/>	whsr03c	whnodec	✖
<input type="checkbox"/>	whsr04a	whnodea	⊘
<input type="checkbox"/>	whsr04b	whnodeb	✖
<input type="checkbox"/>	whsr04c	whnodec	✖

Figure 12-10 Administrative console display of application servers; servers H6SR01A and H6SR04A have unknown status

- We forced a node synchronization and this corrected the server status; see Figure 12-11 on page 310. (One theory is that the Node Agent Mbeans needed to “reinitialize”).

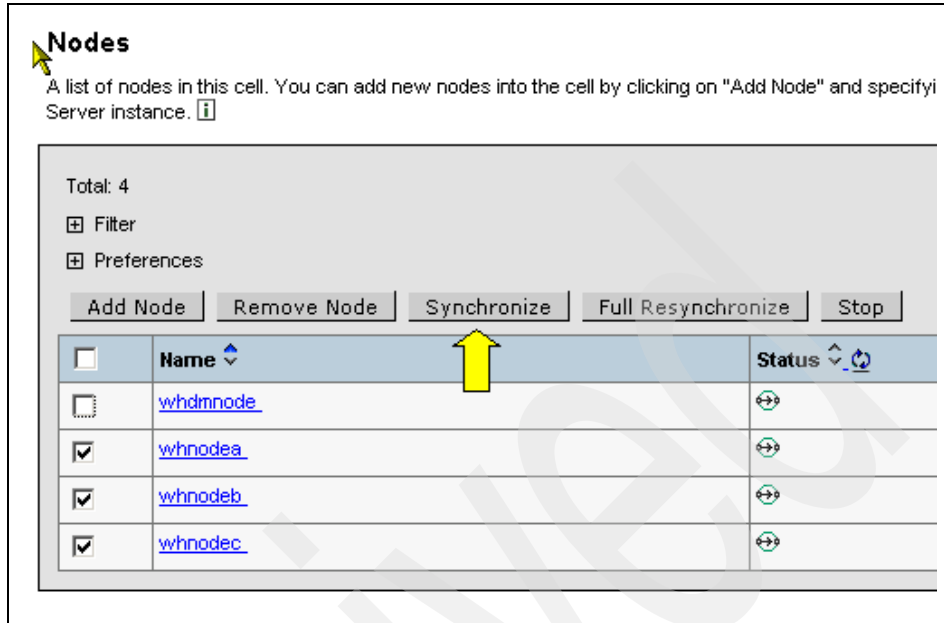


Figure 12-11 Synchronizing nodes to correct server status

12.10 WebSphere control region adjunct

Disabling the control region adjunct will cause all messaging engines in that process to be terminated. However, the server will still be able to accept new HTTP requests, in spite of the failure. The JMS workload, produced by these HTTP requests, will now be redirected to messaging engines in the other servers.

Our test

1. We started our test workload and allowed it to reach full capacity.
2. We killed the adjunct region in server h6sr01a running in LPAR SC49.
3. The application throughput remained the same, because all servers were still available to receive HTTP requests. The messaging engine in server h6sr01b was still available to handle JMS work.

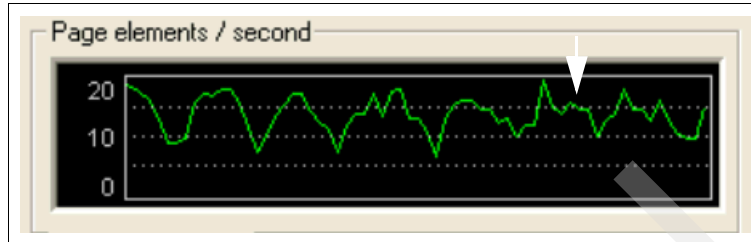


Figure 12-12 WSWS shows no significant drop in throughput

4. The messaging engine failed over to server h6sr01c, which was running in LPAR SC54.
5. When the adjunct region in server h6sr01a was restarted, the messaging engine failed back to that server.
6. The analysis of the JMS workload is shown in Figure 12-13 on page 312. Each column represents the percentage of the work that was consumed and processed by an MDB running in the specified server. Each row represents a sample of 25 responses.

Initially, all JMS work originating in h6sr01a was handled by MDBs in h6sr01a. All JMS work originating in h6sr01b was handled by MDBs in h6sr01b. All JMS work originating in h6sr01c was split between h6sr01a and h6sr01b.

When the CRA in h6sr01a was taken down, the JMS work originating in that server was then sent to h6sr01b, and then split between h6sr01b and h6sr01c, after the messaging engine started in h6sr01c.

When the CRA was restarted in server h6sr01a, the messaging engine was shut down in h6sr01c, and restarted in h6sr01a. The final result was that the pattern of JMS work distribution returned to the way it was before the CRA was taken down.

Displaying message distribution recorded in
zJIBE.Trader_JMSServlet_haplex1_db2.jxs.20050627.1012.log
Press CTRL-C to stop

H6SR01A	H6SR01B	H6SR01C
40%	60%	0%
40%	60%	0%
68%	32%	0%
88%	12%	0%
100%	0%	0%
100%	0%	0%
80%	20%	0%
84%	16%	0%
100%	0%	0%
20%	80%	0% <= took down CRA in h6sr01a
0%	100%	0%
0%	64%	36% <= ME failed over to h6sr01c
0%	60%	40%
0%	88%	12% <= ME failed back to h6sr01a
0%	100%	0%
0%	100%	0%
0%	100%	0%
0%	100%	0%
16%	84%	0%
48%	52%	0%
20%	80%	0%
44%	56%	0%
68%	32%	0%
64%	36%	0%
40%	60%	0%

Figure 12-13 JMS workload distribution across the three servers

12.11 Summary of test results

In all cases, user transactions continued to run without interruption. A few user requests timed out because they were already running on a server we took down. But new requests on that same user session were able to run on another WebSphere application server.

So how can this be proven?

- ▶ Each test script contained a sequence of 20 requests, starting with a logon to establish the session, and ending with a logoff to destroy the session. If HTTP session persistence did not work, then the failure of the application server we were in session with would cause all subsequent requests on that session to fail. The fact that each test showed so few failed HTTP requests indicates that HTTP session persistence was working.
- ▶ Programming logic was added to the test scripts to detect a change in session ID or clone ID while in session, and issue a message. Receiving one of these messages meant that our session had been taken over by another application server. We saw these messages whenever we took down an application server, proving that sessions were failing over to a new application server.

12.12 End-to-end monitoring

By now you may be thinking “How can I monitor the health of all the components in my infrastructure to make sure that they are active and performing well?”

That is a good question, and is especially important when some of the components are running under Linux for zSeries. Linux for zSeries runs under VM, so both VM and Linux need to be monitored.

Knowing that the component has gone down is the first problem, since the redundancy built into our architecture is there to ensure that application workloads continue to run even through multiple component failures. Therefore, you need a way of realizing that a component has failed even though application throughput remains high.

Ensuring the performance and availability of a Web application is not simple. Think of the infrastructure that we created for this redbook. We had a Proxy Server, an Edge Server, and an IHS server all running under Linux under VM in an LPAR. There was another LPAR with a duplicate set of components. Then there were the three z/OS LPARs, each running WebSphere, DB2, CICS, and IMS. All of this needed to be monitored.

We do not address all the monitoring tools in this redbook. For a full discussion of monitoring tools, refer to *Monitoring WebSphere Application Performance on z/OS*, SG24-6825.

Table 12-2 introduces the various products that can be used to monitor components in z/OS, Linux, and networks. We strongly recommend that you deploy a complete monitoring solution when you deploy any Web applications.

Table 12-2 Monitoring tools for z/OS and Linux

Vendor	Product	Use	Overview Web site
BMC	Mainview	Web monitoring and application management.	http://www.bmc.com/products/proddocview/0,2832,19052_19429_22897_1861,00.html
IBM	Omegamon XE	Monitor performance and availability of the Web application server and its Java-based technologies.	http://www.candle.com/www1/cnd/portal/CNDportal_Channel_Master/0,2938,2683_2887,00.htm
IBM	WebSphere Studio Application Monitor	WebSphere Application problem determination and capacity planning for z/OS.	http://www.ibm.com/software/info1/web-sphere/index.jsp
IBM	Tivoli Monitoring for Web Infrastructure	Monitors HTTP and WebSphere on both Linux and z/OS.	http://www.ibm.com/software/tivoli/products/monitor-web/ Tivoli provides products to monitor everything in your infrastructure.
IBM	Tivoli Monitoring for Transaction Performance	Provides a comprehensive display of an end-user transaction's path through your critical business transactions, including response time contributions for each step.	http://www-306.ibm.com/software/tivoli/products/monitor-transaction/
Merrill Assoc.	MXG	Measurement of system resources.	http://www.mxg.com
Sitraka	JProbe	Java performance tuning through application profiling and debugging.	http://www.sitraka.com/software/support/jprobe/tsjprobemainframe.html
Wily	Introscope	Enterprise Web application management for Linux or z/OS. Add-on packages can monitor DB2 and CICS.	http://www.wilytech.com/solutions_ppwebisphere.html

Archived

Archived

Application availability tests

In this chapter we introduce our approach to upgrading a J2EE application within our high-availability WebSphere environment. There are two reasons for upgrading an application:

- ▶ For a planned outage (new version/release of an application)
- ▶ For an unplanned outage (an application error occurred and needs to be fixed)

13.1 New Application Update features

There are two significant improvements in application update:

- ▶ Finer grain application update
- ▶ Rollout application update

13.1.1 Finer grain application update

Previous versions of the WebSphere Application Server for z/OS only supported replacement of an entire application and always stopped and restarted the entire application for any change.

Version 6.0.x introduces an administrative console wizard that can update deployed applications or modules in the following ways:

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

If the application is updated while it is running, the WebSphere Application Server for z/OS automatically stops the application or only its affected components as required, updates the application logic, and restarts the stopped application or its components.

For some components of the application, such as the Web module, EJB module, or a portion of the Web or EJB module, hot deployment and dynamic reloading are also feasible. The containers in WebSphere reload the updated module at specified intervals, without bringing down the application or components.

Finer grain update significantly reduces application downtime and improves application availability.

13.1.2 Rollout Update

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

Rollout Update performs these actions for each cluster member in sequence:

1. Saves the updated application configuration
2. Stops all cluster members on a given node
3. Updates the application on the node by synchronizing the configuration
4. Restarts the stopped cluster members on that node
5. Repeats steps 2,3, and 4 for every node in the cluster one by one until all nodes are updated

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

13.2 Our test

We used the TraderDB application to test application upgrade.

TraderDB includes a Web module and an EJB module. The servlets in the Web module all inherit a servlet called TraderSuperServlet. We modified TraderSuperServlet to test application update.

13.2.1 Rollout Update of the entire TraderDB ear file

To rollout update the entire application, follow these steps below at the Administrative console:

1. Select **Applications** → **Enterprise Application** → Select the application (TraderDBEAR) and click Update; see Figure 13-1 on page 320.

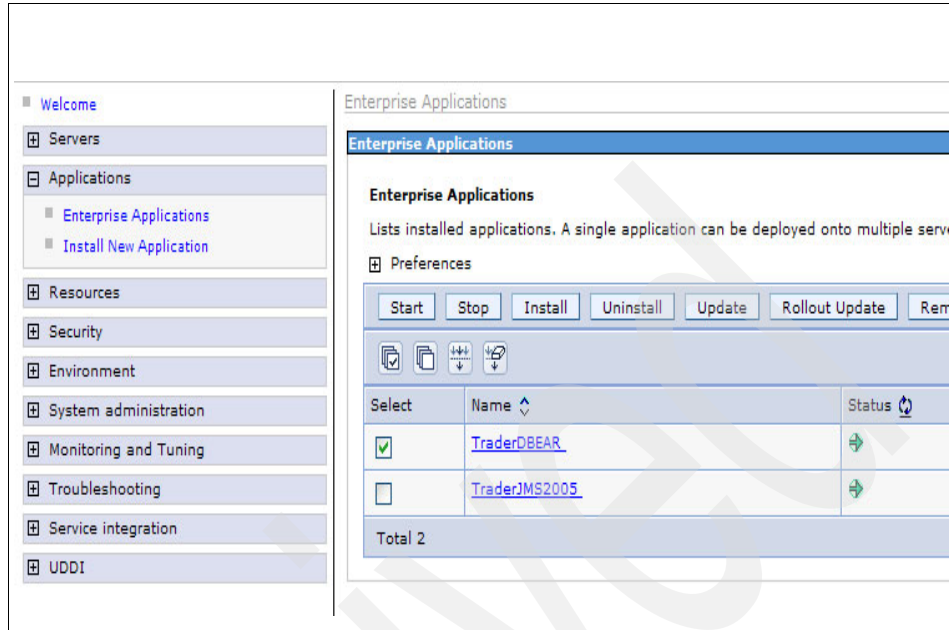


Figure 13-1 Rollout Update TraderDBEAR - step 1

- At the next panel (Preparing for the application installation), select **Full Application** and specify the updated ear file. In this example, the file name is TraderDBEAR_AppUpdate.ear

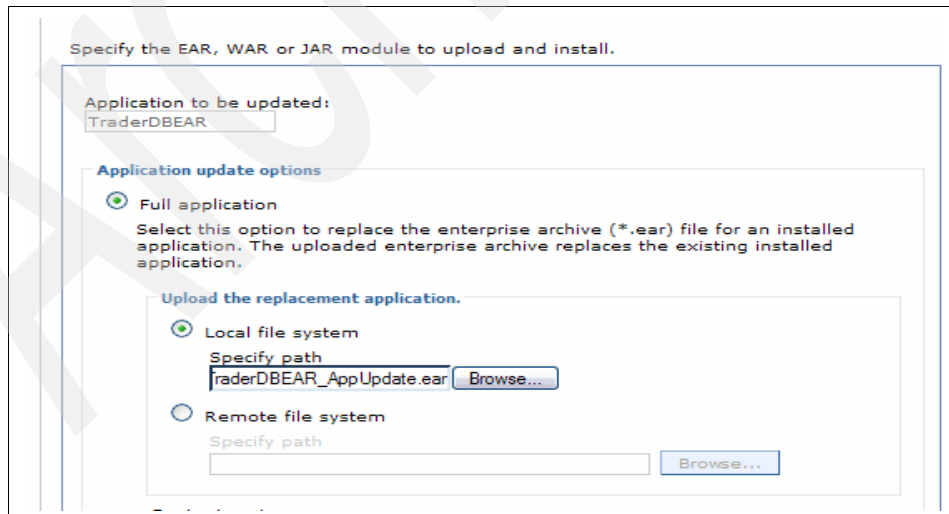


Figure 13-2 Rollout Update TraderDBEAR - step 2

3. Follow the Administrative console wizard steps to update the application (including server mapping, JNDI binding, and so on) just as if you were installing a new application.
4. After finishing the last step of the wizard, the Administrative console will indicate that the application is installed successfully. Now you have three action choices (as shown in Figure 13-3:):
 - **Rollout Update**
This action sequentially updates an application that is installed on multiple cluster members across a cluster.
 - **Save to Master Configuration**
This action save the application update to the master configuration. The update may be immediately available to all server, unless Synchronize changes with Nodes is checked.
 - **Manage Applications**
This action brings you back to the Enterprise Applications panel, to work with installed applications.

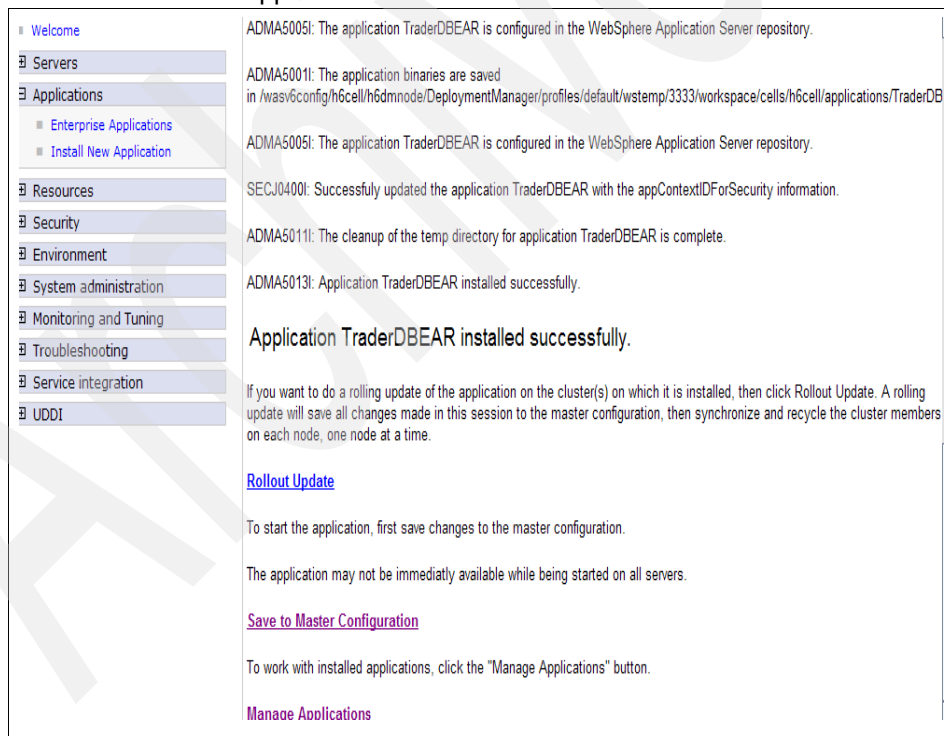


Figure 13-3 Rollout Update TraderDBEAR - step 4

5. Click **Rollout Update**. Now the Administrative console indicates that the Deployment Manager is performing rollout update. Server h6sr01a was first stopped to pick up the update. After server h6sr01a is restarted with the updated application, a second server (h6sr01b) will be stopped and restarted.

However, in our case we experienced a problem with ripple restart of each server. Before server h6sr01a was fully stopped, WebSphere issued a start to it. Because of a lag between the stopping of the servant region and the stopping of the control region, this “restart” failed because the control region being started was already running (although it was in the process of stopping). The control region subsequently took an abend and WebSphere rollout logic waits forever for the failed restart of the server to complete.

We opened a problem for this (PMR - 83871,180,000). To work around this problem, we manually restarted server h6sr01a. After WebSphere noticed that h6sr01a was fully restarted, it continued the rollout update by stopping the second server, h6sr01b. Again the control region took a dump due to the same problem. We manually started server h6sr01b and repeated the process for server h6sr01c.

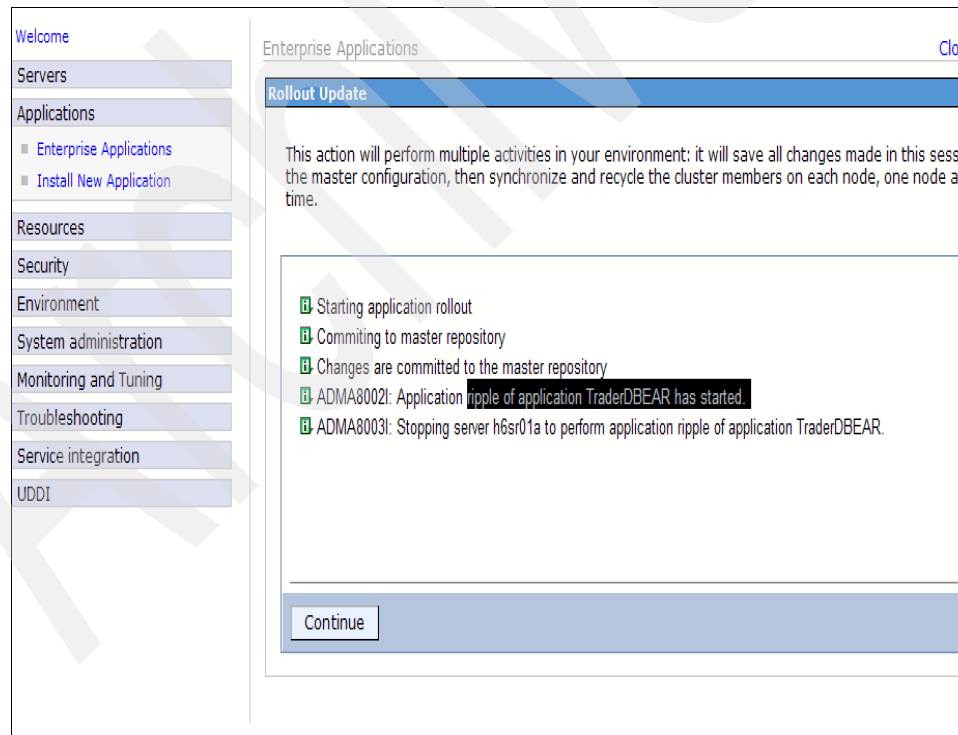


Figure 13-4 Rollout update TraderDB - step 5

6. Subsequently, we noticed that the updated application message <AppUpdate> this is the updated application that we put in the servlet showed up in the servant region sysprint log of all servers. This indicated that the application was fully updated on all servers.

```
customer not exists... create new record  
customer not exists... create new record  
<AppUpdate>this is the updated application
```

Figure 13-5 Rollout update TraderDB - step 6

13.2.2 Single module update

We also tested single module update with the Web module of the traderDB application.

If reloading of application files is enabled and the reload interval is greater than zero (0), then the application's files are reloaded after the application is updated.

For Web modules such as servlets and JavaServer page (JSP) files, a Web container reloads a Web module only when the IBM extension reloading Enabled in the ibm-web-ext.xmi file is also set to true.

You can set reloading Enabled to true when editing your Web module's extended deployment descriptors in an assembly tool or WebSphere Studio Application Developer (WSAD). In WSAD, this property is set at the Extensions tab of Web Deployment Descriptor.

Reloading can also be set at the Admin console for the entire application under **Applications** → **Enterprise Applications** → **application name** → **Class Loading and File Update Detection**.

To update the Web module, follow these steps:

1. Select **Applications Enterprise Application**. Select the application (TraderDBEAR) and click Update.
2. At the next panel (Preparing for the application installation), select **Single module**. Specify the relative path to the module. Because the Web module .war file is a directory under the root of the war file, the relative path is TraderDBWeb.war. Specify the path to the file system to locate the .war file. Also specify the context root, since this is the Web module; refer to Figure 13-6 on page 324.

Specify the EAR, WAR or JAR module to upload and install.

Application to be updated:

Application update options

Full application
 Select this option to replace the enterprise archive (*.ear) file for an installed application. The uploaded enterprise archive replaces the existing installed application.

Single module
 Select this option to update an existing module or to add a new module to the application. If the relative path to the module matches an existing path to a module in the installed application, the uploaded module replaces the existing module. If the relative path to the module does not exist in the installed application, the uploaded module is added to the application.

Relative path to module.

 Path to the existing module, or to the desired path for the new module.

Upload the new or replacement module.

Local file system
 Specify path

Remote file system
 Specify path

Context root
 Used only for standalone Web modules (.war files)

Figure 13-6 Single module update - step 2

3. Follow the steps at the Admin console wizard to update the Web module, including server mapping, EJB mapping and virtual host mapping.

Figure 13-7 on page 325 illustrates server mapping.

Server mapping

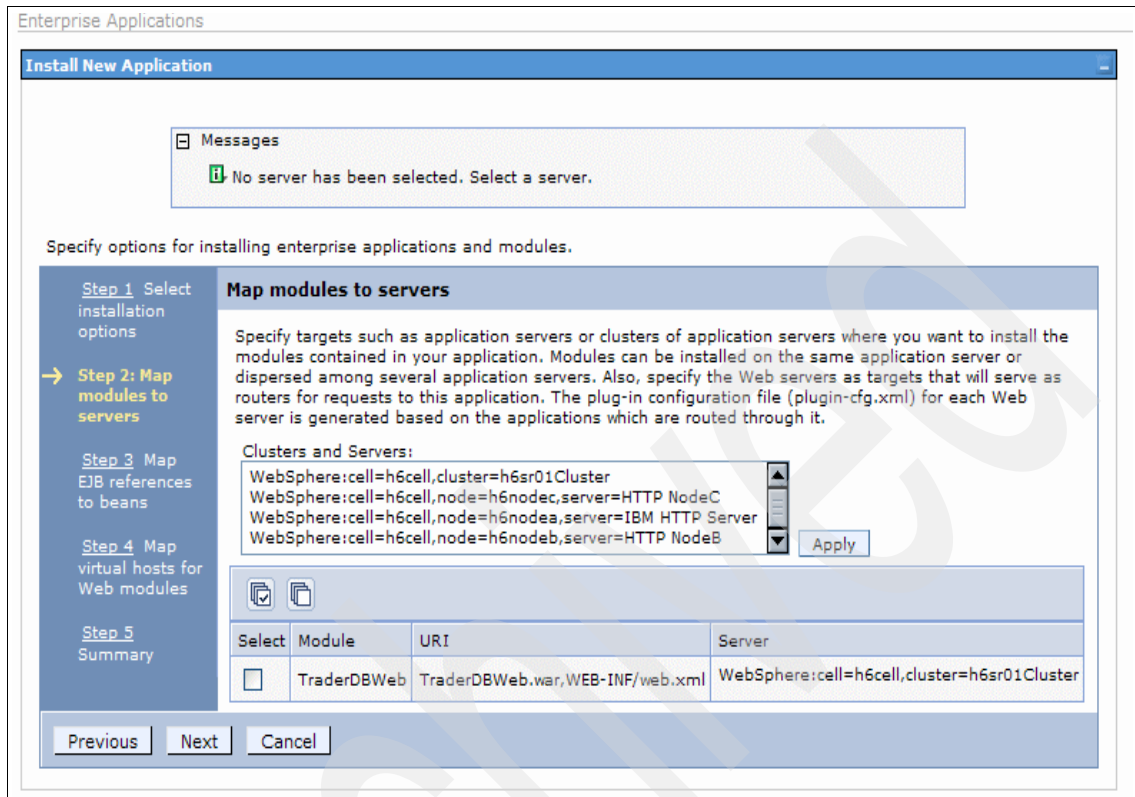


Figure 13-7 Single module update - step 3-1

Figure 13-8 on page 326 illustrates EJB mapping.

EJB mapping

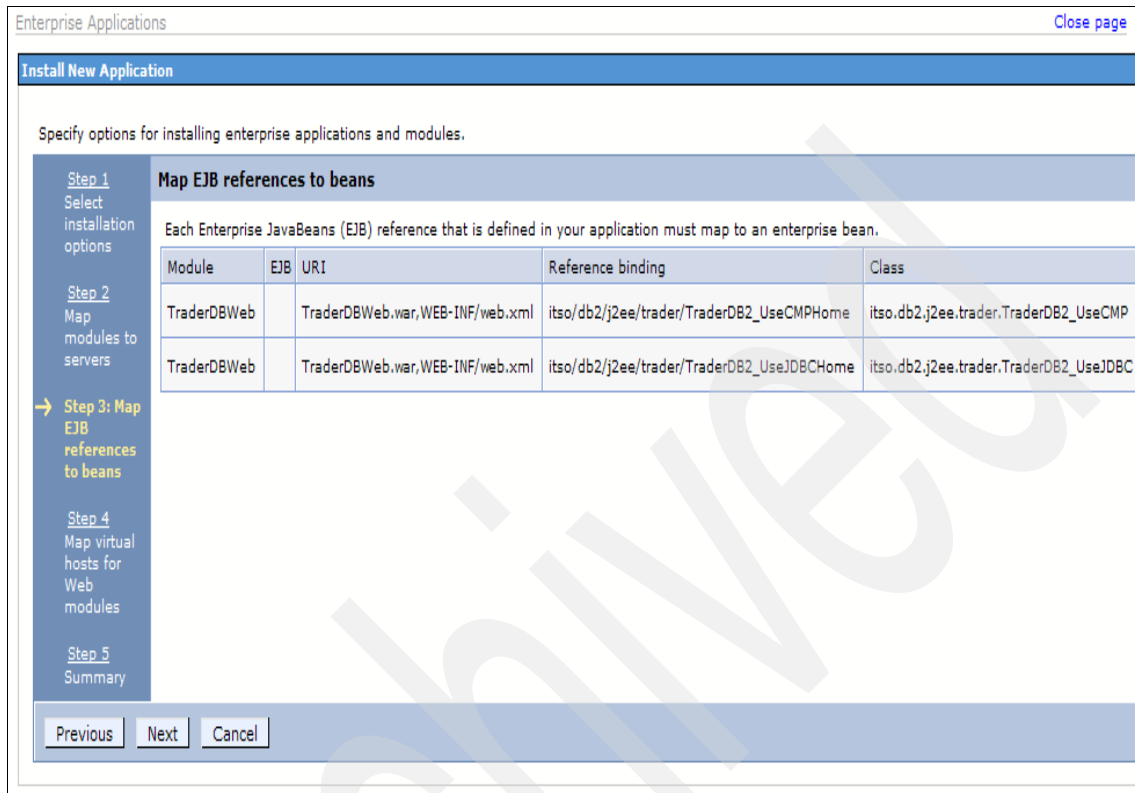


Figure 13-8 Single module update - step 3-2

Figure 13-9 on page 327 continues the illustration.

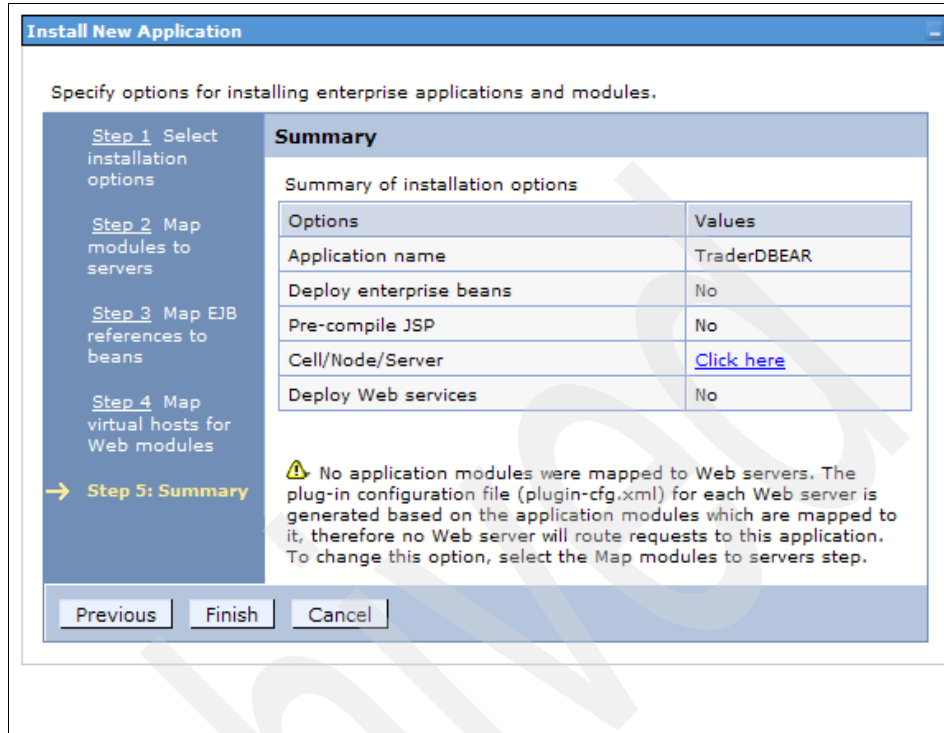


Figure 13-9 Single module update, step 3-3

- After performing the preceding steps, WebSphere updates the .war file to its repository. Select **Manage Application** from the Admin console; refer to Figure 13-10 on page 328.

Updating...

Check the SystemOut.log on the Deployment Manager or server where the application is deployed for specific information about the EJB deployment process as it occurs.

Update of TraderDBEAR has started.

ADMA5009: An application archive is extracted at
/wasv6config/h6cell/h6dmnode/DeploymentManager/profiles/default/wstemp/app_104bf981051/ext/TraderDBWeb.war

ADMA5064: FileMergeTask completed successfully for TraderDBEAR.

ADMA5005: The application TraderDBEAR is configured in the WebSphere Application Server repository.

ADMA5005: The application TraderDBEAR is configured in the WebSphere Application Server repository.

ADMA5005: The application TraderDBEAR is configured in the WebSphere Application Server repository.

ADMA5011: The cleanup of the temp directory for application TraderDBEAR is complete.

Update of TraderDBEAR has ended.

Update of TraderDBEAR has ended.

To start the application, first save changes to the master configuration.

The application may not be immediately available while being started on all servers.

[Save to Master Configuration](#)

To work with installed applications, click the "Manage Applications" button.

[Manage Applications](#)

Figure 13-10 Single module update - step 4

5. Save the configuration change. When the reload time arrives after the update, WebSphere reloads the updated module into the runtime.

13.2.3 Single file update

We also tested a single file update by only updating the servlet class file. Since reload was enabled, there was no need to restart the application after the update. The steps for performing this update are described here:

1. Select **Applications** → **Enterprise Application** → select the application (TraderDBEAR) and click Update.
2. On the next panel (Preparing for the application installation), select **Single file**; refer to Figure 13-11 on page 329.

Application update options

Full application
Select this option to replace the enterprise archive (*.ear) file for an installed application. The uploaded enterprise archive replaces the existing installed application.

Single module
Select this option to update an existing module or to add a new module to the application. If the relative path to the module matches an existing path to a module in the installed application, the uploaded module replaces the existing module. If the relative path to the module does not exist in the installed application, the uploaded module is added to the application.

Single file
Select this option to update an existing file or to add a new file to the application. If the relative path to the file matches an existing path to a file in the installed application, the uploaded file replaces the existing file. If the relative path to the file does not exist in the installed application, the uploaded file is added to the application.

Relative path to file.

Path to the existing file, or to the desired path for the new file.

Upload the new or replacement files.

Local file system
Specify path

Remote file system
Specify path

Figure 13-11 Single file update - step 2

Specify the relative path of the servlet class file in the entire application .ear file (shown in Figure 13-12). Also specify the path to the file system to locate the servlet class file.

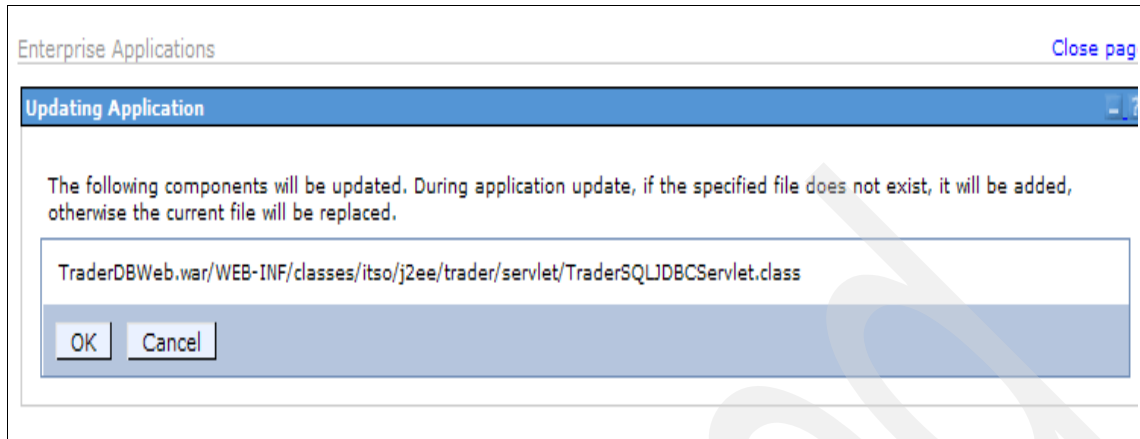


Figure 13-12 Single file update, relative path

3. WebSphere updates the .class file to its repository. Select **Manage Application** from the Admin console.

Updating...

Check the SystemOut.log on the Deployment Manager or server where the application is deployed for specific information about the EJB deployment process as it occurs.

Update of TraderDBEAR has started.

ADMA5009: An application archive is extracted at /wasv6config/h6cell/h6dmnode/DeploymentManager/profiles/default/wstemp/app_104c327c8bf/ext

ADMA5064: FileMergeTask completed successfully for TraderDBEAR.

ADMA5005: The application TraderDBEAR is configured in the WebSphere Application Server repository.

ADMA5005: The application TraderDBEAR is configured in the WebSphere Application Server repository.

ADMA5005: The application TraderDBEAR is configured in the WebSphere Application Server repository.

ADMA5011: The cleanup of the temp directory for application TraderDBEAR is complete.

Update of TraderDBEAR has ended.

Update of TraderDBEAR has ended.

To start the application, first save changes to the master configuration.

The application may not be immediately available while being started on all servers.

[Save to Master Configuration](#)

To work with installed applications, click the "Manage Applications" button.

[Manage Applications](#)

Figure 13-13 Single file update - step 4

4. Save the configuration change. When the reload time arrives after the update, WebSphere reloads the updated module into the runtime.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 336. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Architecting High Availability e-business on IBM eServer zSeries*, SG24-6850
- ▶ *CICS Transaction Gateway V5 The WebSphere Connector*, SG24-6133
- ▶ *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284
- ▶ *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435
- ▶ *IBM eServer zSeries 990 Technical Guide*, SG24-6947
- ▶ *e-Business Cookbook for z/OS Volume 2: Infrastructure*, SG24-5981
- ▶ *zSeries Hipersockets*, SG24-6816
- ▶ *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299
- ▶ *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824
- ▶ *Migrating WebSphere Applications to z/OS*, SG24-6521
- ▶ *Linux for S/390*, SG24-4987
- ▶ *IBM WebSphere V5.0 Performance, Scalability, and High Availability: WebSphere Handbook Series*, SG24-6198
- ▶ *Enterprise JavaBeans for CICS Transaction Server V2.2*, SG24-6284
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *Rational Application Developer V6 Programming Guide*, SG24-6449
- ▶ *WebSphere Application Server V6: System Management & Configuration Handbook*, SG24-6451
- ▶ *More About High-Volume Web Sites*, SG24-6562
- ▶ *WebSphere V5 Security Handbook*, SG24-6573

- ▶ *Enabling High Availability e-Business on zSeries*, SG24-6850
- ▶ *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864
- ▶ *IBM WebSphere V5 Edge of Network Patterns*, SG24-6896
- ▶ *Building Multi-Tier Scenarios for WebSphere Enterprise Applications*, SG24-6956
- ▶ *Migrating Applications from WebSphere for z/OS V4 and V3.5 to V5*, SG24-7044
- ▶ *Tivoli and WebSphere Application Server on z/OS*, SG24-7062
- ▶ *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064
- ▶ *Tivoli Web Solutions: Managing Web Services and Beyond*, SG24-6049
- ▶ *Workload Management for Web Access to CICS*, SG24-6118
- ▶ *Monitoring WebSphere Application Performance on z/OS*, SG24-6825
- ▶ *WebSphere Version 5 Application Development Handbook*, SG24-6993
- ▶ *UNIX System Services z/OS Version 1 Release 4 Implementation*, SG24-7035
- ▶ *Linux on IBM @server zSeries and S/390: System Management*, SG24-6820
- ▶ *IBM WebSphere V4.0 Advanced Edition Handbook*, SG24-6176
- ▶ *WebSphere Edge Server Working with Web Traffic Express and Network Dispatcher*, SG24-6172
- ▶ *IBM WebSphere Edge Server: New Features and Functions in Version 2*, SG24-6511
- ▶ *Linux on IBM zSeries and S/390: High Availability for z/VM and Linux*
- ▶ *Securing Linux for zSeries with a Central z/OS (RACF) LDAP Server*, REDP0221
- ▶ *IBM Tivoli Access Manager for e-business*, REDP3677
- ▶ *Linux on IBM @server zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP3719
- ▶ *Networking Overview for Linux on zSeries*, REDP3901
- ▶ *Linux on IBM zSeries and S/390: High Availability for z/VM and Linux*, REDP-0220
- ▶ *z/VM Configuration for WebSphere Deployments*, REDP-3661

Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS MVS Setting Up a Sysplex*, SA22-7625
- ▶ *IBM WebSphere Application Server for z/OS V5.0.2: Servers and Environment*, GA22-7958
- ▶ *Introduction to WebSphere for z/OS V5*, WP100339
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834
- ▶ *Load Balancer Administration Guide Version 5*, GC09-4602
- ▶ *Load Balancer Administration Guide Version 6.0.1*, GC31-6858
- ▶ *z/VM V4R4.0 TCP/IP Planning and Customization*, SC24-6019
- ▶ *z/VM V4R4.0 General Information*, GC24-5991
- ▶ *z/VM V4R4.0 Running Guest Operating Systems*, SC24-5997
- ▶ *z/VM V4R4.0 Virtual Machine Operation*, SC24-6036
- ▶ *z/VM V4R4.0 System Operation*, SC24-6000
- ▶ *z/OS MVS Setting Up a Sysplex*, SA22-7625
- ▶ *z/OS MVS Programming: Sysplex Services Guide*, SA22-7617
- ▶ *z/OS MVS Setting Up a Sysplex*, SA22-7625
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838
- ▶ *z/OS V1R4.0 MVS Sysplex Services Guide*, SA22-7617
- ▶ *z/OS V1R4.0 MVS Setting Up a Sysplex*, SA22-7625
- ▶ *IBM WebSphere Application Server for z/OS V5.0.2: Applications*, SA22-7959
- ▶ *IBM WebSphere Application Server for z/OS V5.0.2: Resources*, SA22-7960
- ▶ *IBM WebSphere Application Server for z/OS V5.0.2: Performance Tuning and Monitoring*, SA22-7963
- ▶ *z/OS Communications Server: IP Configuration Reference Version 1 Release 4*, SC31-8776
- ▶ *z/OS Communications Server: IP System Administrator's Commands Version 1 Release 4*, SC31-8781
- ▶ *WebSphere Application Server V4.01 for z/OS and OS/390 Operations and Administrations*, SA22-7835-03
- ▶ *SecureWay Security Server: LDAP Server Administration and Use*, SC24-5923

- ▶ *RFC2196: Site Security Handbook*, by B. Fraser
- ▶ *z/VM Security and Integrity*, by Alan Altmark and Cliff Laking
- ▶ *Linux System Security*, by Mann et al.
- ▶

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ You can find a detailed description of the J2EE standard at:
<http://java.sun.com/j2ee/overview.html>
- ▶ WebSphere Edge Components InfoCenter at:
<http://www-306.ibm.com/software/webservers/appserv/ecinfocenter.html>
- ▶ For performance testing results, go to:
<http://www.vm.ibm.com/perf/reports/zvm/html/layer2.html>
- ▶ WebSphere Advanced Edition InfoCenter at:
<http://www-306.ibm.com/software/webservers/appserv/ecinfocenter.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- adding new cluster 209
- address resolution protocol (ARP) 164
- address space isolation 5
- administrative console 28, 170, 219–220, 226, 230, 308–309
- affinity 15
- anatomy, WebSphere scaling 34
- Apache HTTP server 167
- applets 11
- appliance servers, using 42
- application
 - clients 11
 - programming patterns 47
 - server restart 229
 - server system queues, WebSphere 31
 - tuning 29
- applications used 257
- applications used to create the workload 258
- applications, stateless 50
- applying maintenance 233
- architecture, things to consider 100
- architectures for availability 99
- areas, focus 30
- ARM (Automatic Restart Manager) 5
- ARP (address resolution protocol) 164
- audit 112
- authentication 111
- authorization 111
- Automatic Restart Manager (ARM) 5
- availability 54, 100, 119
- avoid failures, parameters to 28

B

- back-end systems 153
- balancing and workload management 21
- batch requests 43
- behavior, user 39
- bindings, configured 48
- bootstrap server 136, 141
- bus 124
- bus members 124
- business model/workload pattern 38

C

- caching 44, 50
- caching proxy 58, 60, 62
- cells 35
- changes to configuration, dynamic 26
- CICS 24, 106, 109–110, 114–116, 118, 257, 313
- Cisco 53, 66–68
- Cisco Workload Agent for OS/390 67
- CloneID 87, 89, 169, 173, 218, 263, 270
- cluster address 54, 86, 120, 158, 161–164, 172, 218
- cluster of systems, creating 42
- clusters and cluster members 34
- configuration
 - dynamic changes to 26
 - HFS structure 188
 - mixed 26
- configured bindings 48
- configuring
 - network adapter interfaces/cluster IPs 157
 - Network Dispatcher to load on reboot 159
 - the WebSphere plug-in 169
- connection dispatching 58, 66
- connection optimization (DNS/WLM) 55, 57
- connections, managing 43
- connectors, J2EE 50
- considerations for scalability 19
- console, administrative 28
- content-based forwarding 62
- content-based routing 62
- context root 174
- control process 304
- control region 9, 87, 89, 298, 301
- cookie 72, 74, 87–89, 94, 121, 169–170, 173–174, 214–216, 263, 305
- cost 101
- creating a cluster of systems 42

D

- data store 127
- data-sharing 5
- DB2 16, 24, 94, 108–110, 113–114, 117, 155, 225–226, 231, 255, 257–258, 263, 313

- DB2 datasource setup 225
- DB2 Idle Thread Timeout 242
- decouple
 - namespace lookups 121
 - Web and EJB layers 118
 - Web/EJB and EIS layers 113
- default messaging provider 9, 124, 136
- defining default virtual host 214
- defining the cluster 208
- definitions
 - horizontal scalability 20
 - performance 20
 - scalability 20
 - vertical scalability 20
- deployment descriptor 12
- deployment manager 121, 188, 229, 299, 307–308
- deployment manager with Peer Recovery 307
- destinations 128
- distributed DVIPA 63
- distributing stack 63
- distribution manager 58
- distributor, sysplex 41
- DMZ 61
- DNS 55–57, 63, 170, 213
- DNS (Domain Name Server) 55, 173
- DNS mapping 55
- DNS/WLM 57
- Domain Name Server (DNS) 55, 173
- Domain Name Server mapping solutions 55
- DVIPA 65, 86, 108, 114, 117, 180, 184
- dynamic changes to configuration 26
- dynamic VIPA (DVIPA) 63, 178

E

- Edge Server 44, 62, 85, 108, 120, 168, 175, 217, 299, 313
- EIS (Enterprise Integration Subsystems) 105
- EJB 257
- EJB handle 96
- ejbCreate 96
- enable cookies 214
- end-to-end monitoring 313
- Enterprise Integration Subsystems (EIS) 105
- Enterprise JavaBeans 11
- entity beans 12
- environment, scaling your 33
- example to roll the upgrade 286

F

- failover 125, 130, 133, 253
- failover 22
- failures, parameters to avoid 28
- faster system, using a 41
- fault isolation 26
- firewall 41–43, 61, 101, 107, 176
- foreign bus 130
- front-end systems 151

G

- GDPS (Geographically Dispersed Parallel Sysplex) 5
- Generic Routing Encapsulation (GRE) 60, 62
- Generic Routing Encapsulation (GRE) feature 167
- Geographically Dispersed Parallel Sysplex (GDPS) 5
- GRE (Generic Routing Encapsulation) 167

H

- hardware-based high availability 22
- HFS 187–188, 232, 234, 286, 289–291, 296
- HFS setup for SMP/E environment 186
- HFS structure for upgrades 232
- HFS versus zFS 234
- high availability, hardware-based 22
- high availability 21
- high availability, hardware-based 22
- horizontal scaling 37
- host name 54–55, 174
- host name resolution 57
- hotlist, tuning parameter 28
- how the definitions fit together 217
- how the traffic flows 174
- HTML 12
- HTTP
 - server plug-in 121
 - session 94, 214
 - session affinity 15
 - Transport Handler 87, 169
- HTTP advisor 161
- HTTP server 28, 86, 88, 107–108, 111, 167–168, 301
- HttpSession object 96
- HVWS simulator usage 41

I

- IBM HTTP Server (IHS) 167
- IBM HTTP Server configuration (optional) 168
- IBM HTTP Server on Linux for zSeries 167
- IHS 167, 214, 217
- IIOF 94
- IIOF workflow in a parallel sysplex 64
- implementation we used 179
- IMS 24, 106, 109, 116, 258–259, 313
- increasing performance 27
- increasing throughput 44
- in-doubt transactions 132
- in-flight transactions 132
- infrastructure availability tests 285, 297
- infrastructure considerations 150
- infrastructure model 40
- infrastructure, tune the 30
- installing new maintenance levels of WebSphere for z/OS 286
- Intelligent Resource Director (IRD) 7
- interfaces, remote and local 49
- interfaces, strongly typed 49
- interoperability 121
- IP subnet 61
- IRD (Intelligent Resource Director) 7
- isolation, fault 26

J

- J2C resources security 112
- J2C resources 110
- J2EE 10
- J2EE connectors 50
- J2EE server 87
- Java 2 Enterprise Edition 12
- JavaServer Pages 11
- JSESSIONID 87, 89, 121, 168, 170, 173, 214
- JSESSIONID cookie 168

L

- Linux 6, 60, 119, 147, 150, 156, 164–165, 176, 299, 314
- Linux for zSeries 313
 - kernel modules update 156
 - what's going on with ARP? 165
- Linux guest 299
- load balancer 38, 58–60, 161, 176
 - configuration 161
 - high availability configuration 161

- load balancing 14
- load balancing and workload management 21
- local interfaces, remote and 49
- local performance vs. workload balancing 112
- logical architectures 102
- logical tiers 101
- LPA 292
- LPAR 178, 301, 304, 308

M

- MAC (Media Access Control) 61
- MAC forwarding 60–61, 88, 164
- maintainability 25
- maintenance level 231, 234
- maintenance strategy 231
- management, load balancing and workload 21
- management, session 26
- manager, deployment 35
- manager, workload 67
- managing connections 43
- measurement, performance 28
- Media Access Control (MAC) 61
- mediation 129, 144
- mediation point 129
- message-driven beans 12
- messaging engine 125
- mixed configuration 26
- MNLB (MultiNode Load Balancing) 58, 66
- model/workload pattern, business 38
- model, infrastructure 40
- monitoring 27–28, 60, 163, 314
- monitoring tools 314
- MultiNode Load Balancing (MNLB) 58, 66

N

- naming convention 189
- NAT (Network Address Translation) 60
- NAT/NAPT forwarding 61
- ND
 - active 162
 - auto recovery 162
 - backup 162
 - heartbeat monitor 162
 - loopback device 162
 - manual recovery 162
 - primary 162
 - recovery strategy 162
 - script

- goActive 163
- goInOp 163
- goStandby 163
- standby 162
- Network Address Translation (NAT) 60
- Network Address Translation (NAT/NAPT) forwarding 61
- Network Dispatcher 59–62, 85, 151, 156, 158–159, 161, 217, 299
 - installation 156
 - methods to configure 159
- network interface aliasing 164

O

- outage, planned 22
- overview, caching 46

P

- Parallel Sysplex 5, 185
- parameter hotlist, tuning 28
- parameters to avoid failures 28
- partition 129
- patterns 47
 - application programming 47
 - business delegate 47
 - business delegate factory 49
 - business model/workload 38
 - data transfer object 48
 - other considerations 48
 - service locator 48
 - session façade 47
- Peer Recovery 23, 219, 298
- Peer Restart and Recovery (PRR) 24, 228
- performance
 - definition 27
 - increasing 27
 - measurement 28
 - tuning 29
 - advanced techniques 30
 - focus areas 30
 - techniques, advanced 30
 - techniques, scaling 41
- physical architectures 104
- planned outage 5, 22, 317
- plug-in installation 168
- plugin-cfg.xml 170, 173
- port numbers 56, 194
- process, scaling 35

- programming patterns, application 47
- provider endpoints 136
- Proxy server 59, 62, 300
- PRR (Peer Restart and Recovery) 228
- PTF 114, 186, 289
- publication point 129
- publish/subscribe 129
- pull 135
- push 135

Q

- queue point 128
- queues, WebSphere Application Server system 31
- queuing before WebSphere 33

R

- recovery services 5
- recovery, peer 23
- recovery 24
- Redbooks Web site 336
 - Contact us xiii
- remote access to
 - CICS 114
 - DB2 113
- remote and local interfaces 49
- remote connection mechanisms 113
- requests, batch 43
- Resource Recovery Services (RRS) 5, 24
- RMI 94
- rolling upgrade process 289
- round-robin DNS 55
- router 66
- RRS 5, 24, 110, 155, 226, 255
- RRS (Resource Recovery Services) 5, 24

S

- scalability 100
- scaling
 - anatomy, WebSphere 34
 - considerations 19
 - horizontal 37
 - process 35
 - techniques 41
 - the environment 33
 - vertical 35
- script
 - Trader_db2 263

- scripts for WebSphere Studio Workload Simulator 263
- SD distributing stack 179
- security 27, 111, 122
- segmenting the workload 43
- self-healing 5
- serializable 16
- servant process 304
- server
 - group name 57
 - HTTP 28, 38, 46
 - instance 89
 - region 87, 301
- Server, Edge 44
- servers, using appliance 42
- service integration bus 9, 124, 130
- service name 54–55
- servlet 11, 94
- session
 - affinity 15–16, 73, 75–76, 86, 167, 169
 - cookie 86–87
 - ID 94
 - invalidation
 - timer 89
 - management 26
 - changes 226
 - object 87
 - persistence 16, 85, 225
- setting up the infrastructure 149, 177, 237
- setting up WebSphere 185
- setup for memory-to-memory internal replication 219
- setup for session persistence with DB2 225
- SFM (Sysplex Failure Management) 230
- simulator usage, HVWS 41
- single system image 63
- SMP/E 286, 291
- stateful session bean 95
- stateless applications 50
- stateless EJB 258
- stateless protocol 13
- StoneGate 123
- storage key protection 5
- strongly typed interfaces 49
- subnet 61
- summary of test results 313
- summary of unplanned outage tests 298
- sysplex 57, 286
- sysplex distributor 63–65, 86, 89, 114, 117, 155,

- 170, 172, 178–179, 299
- sysplex distributor and MultiNode Load Balancing (MNLB) 66
- Sysplex Failure Management (SFM) 230
- system queues, WebSphere Application Server 31
- systems, creating a cluster of 42
- system-wide identity 54

T

- target infrastructure 186
- TCP/IP 59, 64, 110, 155, 181–182, 287, 306
- TCP/IP packet header 62
- three-tier
 - EIS decoupled with high availability 115
 - logical architecture 103
 - physical architectures 112
- throughput 20
- throughput, increasing 44
- time-to-live (TTL) 56
- Trader 260, 263
- TraderDB 258
- TraderDB2 258
- TraderJMS 258
- transaction 12
- transactions, in-doubt 132
- transactions, in-flight 132
- TTL (time-to-live) 56
- tuning
 - parameter hotlist 28
 - performance 29
 - the application 29
 - the infrastructure 30
- two-tier
 - logical architecture 102
 - physical architecture 106
 - with high availability 108

U

- unavailable 132
- understanding your workload 38
- UNIX System Services 260
- unplanned outage 154, 298, 317
- URL rewriting 94
- usage, HVWS simulator 41
- user behavior 39
- using a faster system 41
- using appliance servers 42

V

- verify and configure
 - dynamic cache service 223
 - session management 220
 - the internal replication domain 219
- vertical scaling 35
- VIPA 63, 172, 178, 288, 299
- VIPA (Virtual IP Address) 63
- Virtual IP Address (VIP) 63

W

- Web components 12
- Web container 169, 213
- Web container definitions 213
- Web server forwarding configuration 164
- webcontainer.conf 89
- WebSphere
 - configuration 188
 - flow of installation 188
 - libraries setup 187
 - procedures 195
 - queuing 33
 - scaling anatomy 34
 - servant process 302
 - server 34
- WebSphere Advanced Edition Web server plug-in 168, 217
- WebSphere Application Server system queues 31
- WebSphere Edge Components/Web server plug-in 168
- WebSphere Edge Server 62, 217, 299–300
- WebSphere MQ 105, 124, 130
- WebSphere Studio Workload Simulator 260, 303–304
- WLM 57
- WLM GOAL mode 178
- workload 262
 - balancing 53–54
 - base components 38
 - distribution mechanisms 84
 - management, load balancing 21
 - segmenting 43
 - understanding 38
- Workload Manager 9, 67, 178

Z

- z/OS base infrastructure 105



Architecting High Availability Using WebSphere V6 on z/OS

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Architecting High Availability Using WebSphere V6 on z/OS

Theory and definitions

Linux for zSeries and z/OS working together

High-availability scenarios

In this IBM Redbook, we describe how to configure the various components of an e-business solution to exploit the availability and scalability benefits of zSeries and Parallel Sysplex using multiple LPARs running Linux for zSeries and z/OS.

This publication applies to WebSphere for z/OS V6, and is a continuation of the project evaluating high availability using WebSphere for z/OS V5 and V4. Presented in three parts (theory, systems setup, and availability tests), we cover workload balancing, the use of HTTP sessions, various architectures, and how to set up and test your infrastructures.

Considerations for configuring the systems and applications in an e-business environment are also examined, and we address such questions as:

- ▶ Should there be separate front-end systems for running the WebSphere Application Server?
- ▶ How should e-business components be configured to minimize or eliminate the impact of a system outage?
- ▶ What is the best way to communicate between the WebSphere systems and the back-end application systems?
- ▶ What is the impact on the end user if there is a failure in one of the front-end or back-end systems?

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks