

IBM Washington Systems Center

WebSphere Application Server V4.0.1 for zOS and OS/390

Establishing Test and Production Environments

Key Issues and Considerations

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number WP100266 under the category of "White Papers"

Version Date: August 27, 2002

(See "Change History" on page 39 for description of modifications to this document)

Donald C. Bagwell
IBM Washington Systems Center
1-301-240-3016
dbagwell@us.ibm.com

Robert H. Teichman
IBM Washington Systems Center
1-301-240-8616
teichmn@us.ibm.com

The following people were instrumental in the construction of this white paper:

- Mike Cox, Washington Systems Center
- John Hutchinson, Washington Systems Center

Table of Contents

Background on the Issue	1
Before you get started: a key assumption used in the writing of this document	1
How to read this document.....	1
What is meant by "Isolation"	1
A note about application testing versus system software testing	2
Illustration of some important concepts	3
Servlets and the WebSphere V3.5 Environment	3
The WebSphere for zOS Customization ISPF dialog	5
WAS configuration information and where it is stored	5
Configuration data and application data stored in the HFS	6
WAS "nodes"	8
Question: Is it possible to have more than one WAS node in a sysplex?	8
Question: Is it possible to have more than one WAS node within a system or LPAR?	9
Sharing within a sysplex.....	9
The Systems Management End User Interface (SMS EUI)	10
Single WAS node test/production configuration	11
Multiple WAS node test/production configuration.....	11
The WAS naming convention	12
Adding J2EE application servers after initial configuration	12
JNDI names applied to deployed applications	13
Question: what exactly is that long "default JNDI name" string SMS EUI sets?	14
Application fallback and version control	15
Multiple Environment Configurations	16
Overview of the four configurations	16
Conf #1 - Different WAS J2EE Servers on the same system or LPAR	17
Snapshot rating of configuration	17
Design Assumptions.....	17
Detail on "Ease of Configuration".....	17
Detail on " Ability to minimize resource usage"	17
Detail on "Degree of <i>application</i> programming isolation"	17
JNDI Names.....	18
J2EE Resource Names.....	19
Web application virtual host and context root settings	20
Detail on "Degree of <i>system</i> programming isolation"	21
Detail on "Degree of human isolation"	21
Conf #2 - Different WAS J2EE servers on different systems in the same WAS node	22
Snapshot rating of configuration	22
Design Assumptions.....	22
Detail on "Ease of Configuration".....	22
Detail on " Ability to minimize resource usage"	23
Detail on "Degree of <i>application</i> programming isolation"	23
Detail on "Degree of <i>system</i> programming isolation"	23
Detail on "Degree of human isolation"	24
Conf #3 - Different WAS nodes within the same Sysplex	25
Snapshot rating of configuration	25
Design Assumptions.....	25
Detail on "Ease of Configuration".....	26
Understand the things you'll code differently in the ISPF customization dialogs.....	26
Planning your server names	27
SMS Administration ID.....	28
TCP Ports and IP names	28

WebSphere V4.0.1 – Test and Production Environments

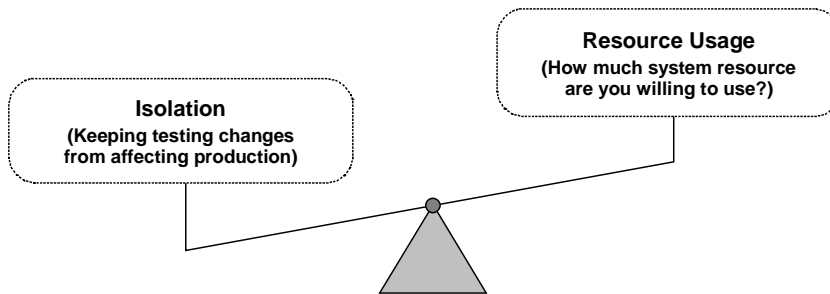
<i>Datasharing environment variable and the bootstrap</i>	29
<i>Configuration using the ISPF dialogs</i>	30
<i>Authorization to backend data stores</i>	30
Detail on "Ability to minimize resource usage"	31
Detail on "Degree of <i>application</i> programming isolation"	32
<i>JNDI names</i>	32
<i>Data resource names defined to WAS in each node</i>	33
<i>Web application virtual host and context root values</i>	33
Detail on "Degree of <i>system</i> programming isolation"	33
<i>What is shared in this configuration</i>	34
Detail on "Degree of human isolation"	34
Conf #4 - Different WAS nodes in separate sysplexes	35
Snapshot rating of configuration	35
Design Assumptions.....	35
Detail on "Ease of Configuration".....	36
Detail on " Ability to minimize resource usage"	36
Detail on "Degree of <i>application</i> programming isolation"	36
Detail on "Degree of <i>system</i> programming isolation"	36
Detail on "Degree of human isolation"	36
Appendix A: Naming Checklist	37
Common Definitions.....	37
Base Servers	37
LDAP Server	38
Each J2EE Application Server Created	38
Change History of Document	39
Index	40

Background on the Issue

Virtually all customers maintain some form of "Test" and "Production" (or perhaps more) environments for their systems and applications. The basic reason for this is to isolate one from the other so that testing doesn't hurt or impact production. Only when changes have been tested and validated are they then moved to the production environment.

Those customers who have adopted WebSphere Application Server Version 4.0.1 for zOS and OS/390 face the task of configuring the same kind of environment for that product as well. So the issue is this: "How do I configure a WAS 4.0.1 "test" and "production" environment, and what are the things I should be aware of as I plan for this?"

Like many things, there is no one "right" answer. This is a trade-off between system isolation and resource usage. The desire is to maximize the degree of isolation while using the minimum amount of resource. The reality is that with isolation comes cost. So the quest is to find the balance that serves your purpose.



The balance between isolation and resource cost

Note: From this point forward the topic will be made simpler by assuming just two environments: "test" and "production." Some may wish to have three or perhaps four or more environments. The key is moving from one environment to *more than one*. Once you're past a single environment, the concepts are pretty much the same whether you have two, three or twenty.

Before you get started: a key assumption used in the writing of this document

This document was written based on the assumption that the reader has at least *some* familiarity with the way in which WAS is installed and configured. *This document is not trying to be a step-by-step "how to" manual on the installation process.* It is designed to provide you with things you should be aware of as you configure a test and production environment.

How to read this document

The section titled "Illustration of some important concepts" on page 3 should be read by all readers of this document. That section lays the foundation for much of what's discussed elsewhere. Then go to "Multiple Environment Configurations" on page 16 to see which configuration best suits your needs.

What is meant by "Isolation"

The reason why changes aren't made directly to the production environment without first testing is to avoid having something suddenly stop working. When setting up a "test" and "production" environment you strive to keep changes made to the "test" environment from impacting the "production" environment. This is "isolation," and it comes in two forms:

- **Programming, or code isolation** – this is making certain that a fix, or an updated piece of code, when installed in the test environment, doesn't somehow get applied to the

WebSphere V4.0.1 – Test and Production Environments

production environment. This topic applies to both system programming and application programming. WAS has many shared resources, and not understanding how they relate to one another can cause problems.

- **Human, or organizational isolation** – this is making certain that a person doesn't have the ability to accidentally or on purpose go in and make changes to things you don't want them to have access to.

The different configurations have varying degrees of each type of isolation. This document will explain the degree of isolation each provides, and explain it in some level of detail.

A note about application testing versus system software testing

This document will focus primarily on the topic of *application testing*; that is, the validation of application code in the form of J2EE applications and association web applications, and the migration of that application code from the test environment to the production environment.

The issue of system software testing is important. But WebSphere for zOS and OS/390 is not that much different from any other subsystem when it comes to maintaining separate target libraries, link libraries, etc.

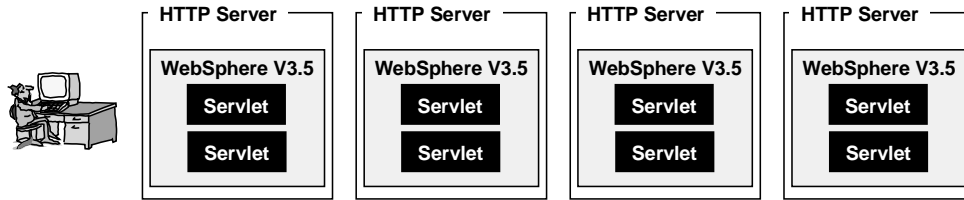
This document will make reference to issues related to system software testing when such a reference is significant. But system software testing will not be a primary focus of the document.

Illustration of some important concepts

To understand the issues that are discussed throughout this document, it's handy to have a few key concepts under your belt.

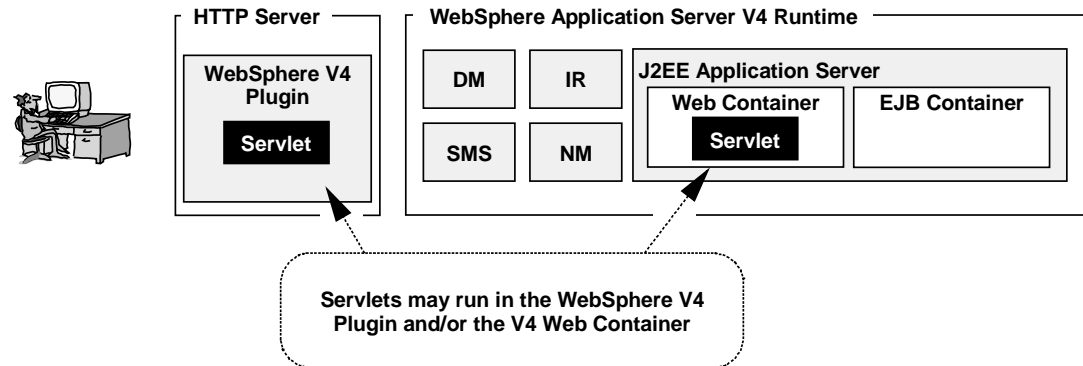
Servlets and the WebSphere V3.5 Environment

Before we start the discussion on Test and Production environments, it is important to understand the difference between running servlets and running EJBs. Many people have servlets running today in a WebSphere Application Server V3.5 environment. In that environment, multiple environments can be created quite easily simply by setting up multiple HTTP Servers:



Multiple HTTP Servers permitted multiple WAS V3.5 environments

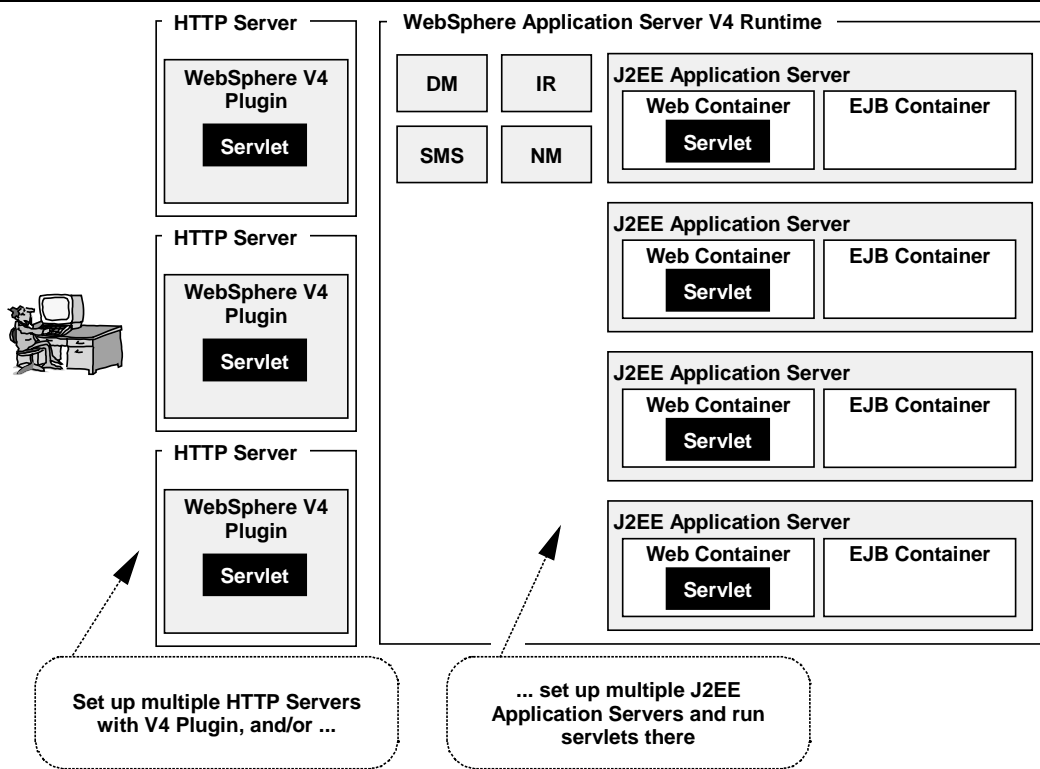
In the WebSphere V4 environment, there are two places where servlets can be run – in the V4 "Plugin" or the "Web Container:"



Two places where servlets can be run in WebSphere Application Server V4

So the question often comes up: "Is it possible to have multiple servlet environments like we had in WebSphere V3.5?" And the answer is "yes," but in two different ways: by setting up multiple Plugins or setting up multiple J2EE application servers:

WebSphere V4.0.1 – Test and Production Environments

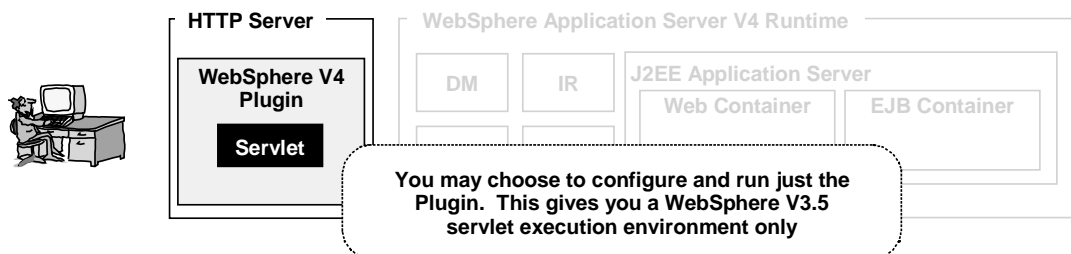


Multiple servlet execution environments in WebSphere V4

The issues surrounding multiple V4 Plugin environments is the same as it was with the WebSphere V3.5 plugin: you simply set up multiple HTTP Servers. The issues surrounding multiple J2EE application servers is different and is covered throughout this document.

Note: If you have *just servlets* and no EJBs, the topic of "hard coded JNDI references" and "unique JNDI names," mentioned throughout this document, doesn't really apply. However, that topic *does apply* if your servlets *call EJBs*: depending on how the servlet is written, it might reference the EJB with a hard-coded reference. If so, then the limitations of a shared JNDI namespace in Configuration #1 and #2 of this document definitely come into play.

The next question that frequently comes up is: "Is it possible to configure and run just the Plugin, and *not* make use of the rest of the runtime environment?" And the answer to that is "yes." This is known by some as the "simple configuration" or "alternative configuration." (You still need to purchase the whole WebSphere V4 license, however.)



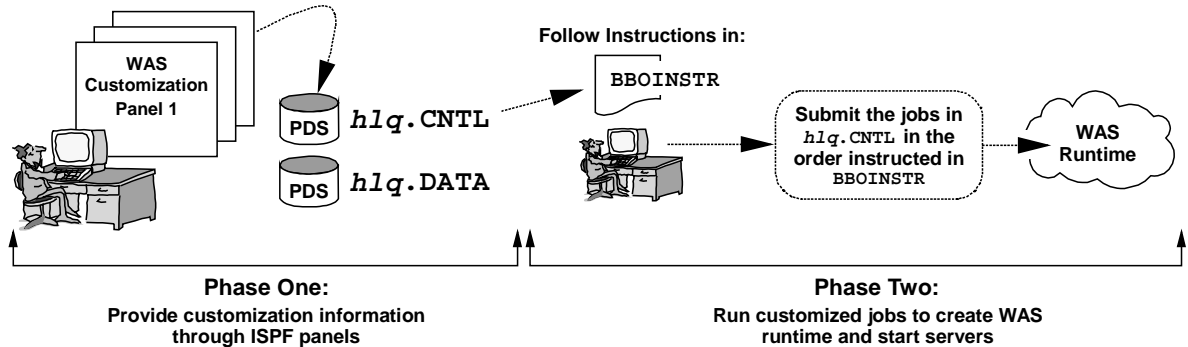
Running just the WebSphere V4 Plugin and not the rest of the runtime is possible

Please refer to "Configuring Web Applications" (WP100238 found in the white paper section of www.ibm.com/support/techdocs) for information on configuring the V4 Plugin.

The WebSphere for zOS Customization ISPF dialog

The creation of a WAS runtime – be it your first or a subsequent runtime for test – is done using what's commonly known as the "WAS ISPF customization dialogs." The customization dialog is a set of ISPF panels in which you enter information about your system and information about the WAS runtime you wish to create. The output from the dialog is a set of JCL jobs, RACF scripts and instructions that have been customized with the information you provided in the panels.

The runtime is not created by the ISPF customization dialog. The runtime is created when you submit the various jobs created by the ISPF program:



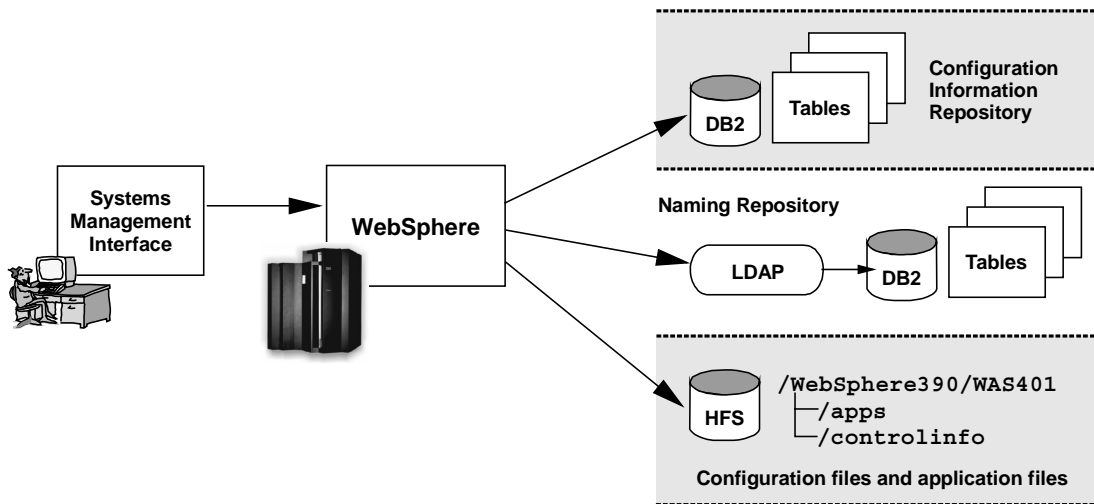
High-level view of the WAS ISPF customization dialog

The ISPF customization dialog panels are a key part of the installation and customization process. This document will make reference to the the ISPF dialogs frequently.

Note: After you get your WAS runtime established you'll eventually want to create additional J2EE servers for your applications. That process falls outside the ISPF customization dialogs and is covered under "Adding J2EE application servers after initial configuration" on page 12.

WAS configuration information and where it is stored

When you install WAS 4.0.1, create a new application server, install a J2EE application, or define some other resource to WAS, WAS will turn and store the information in several different places, depending on what's being stored:



Three places where WAS stores information

WebSphere V4.0.1 – Test and Production Environments

- **DB2** – WAS maintains nearly 100 DB2 tables in which it stores all sorts of information about the configuration of the WAS node, the various servers and the applications that are installed.
- **LDAP** – WAS stores object naming information in LDAP, which in turn stores the information into its own set of DB2 tables.
- **HFS** – WAS maintains a fairly extensive directory structure in which it stores the application files (class files, webapp files), and the configuration and environment variable files for the servers and resources.

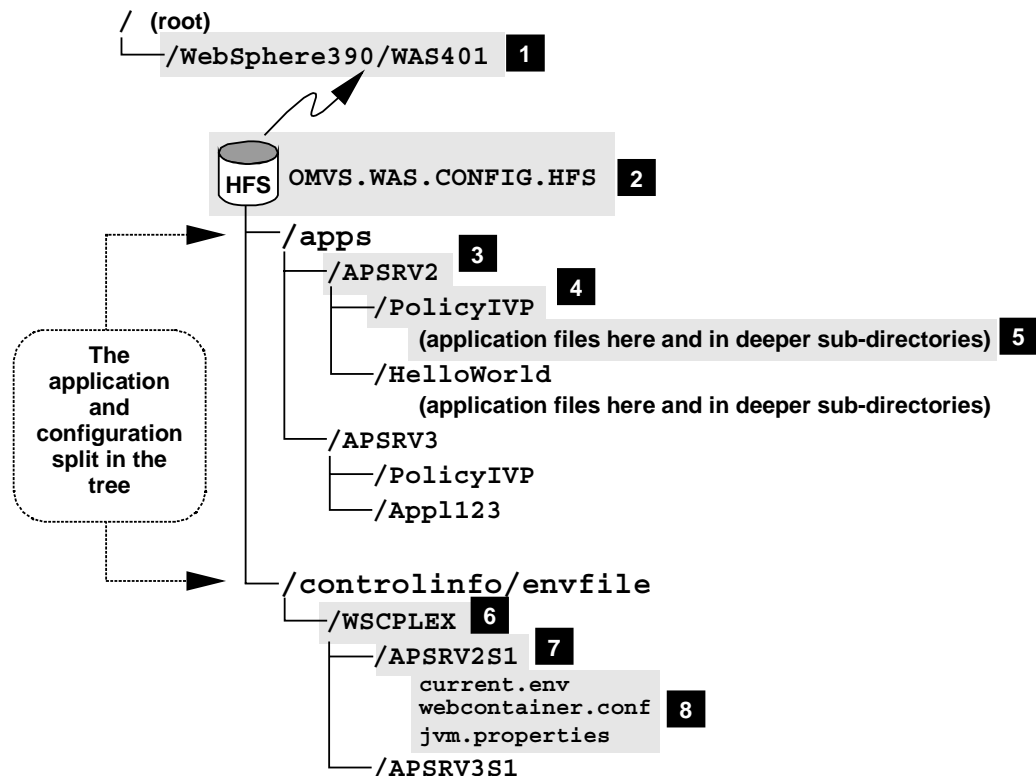
The message here is this: WAS stores its information in more than just one place. This becomes important when the topic of isolation and change migration comes up. These storage locations are shared within a given WAS "node" (see "WAS "nodes"" on page 12 for a definition of this term), and that WAS node may span systems in a sysplex. A change made on one system or LPAR may very well affect other systems as well.

Configuration data and application data stored in the HFS

There are two HFS storage locations of interest when it comes to J2EE applications deployed into a WAS 4.0.1 J2EE application server:

- Location where the server instance configuration data is maintained
- Location where the deployed applications are kept (EJBs and WebApps)

The following picture illustrates the heirarchy:



Where application and configuration information is stored in the WAS 4.0.1 HFS

The numbered blocks are discussed next:

WebSphere V4.0.1 – Test and Production Environments

- 1 The HFS mount point on which the WAS 4.0.1 HFS is mounted is specified in the ISPF customization dialogs under the first panel of "WebSphere Customization," under the heading of "WebSphere HFS Information" and in the field labeled "Mount Point." This is also referenced by the `CBCONFIG=` symbolic in the JCL start procs for the control region and server region of the J2EE servers.

Note: if your WAS node spans systems in a sysplex, this HFS is shared between the systems.

- 2 The data set that contains the HFS file system is specified in the ISPF customization dialogs under the first panel of "WebSphere Customization," under the heading of "WebSphere HFS Information" and in the field labeled "Name."
- 3 Under the `/apps` directory you will find directories with the same name as the J2EE servers you have created. This will be the server name, *not* the server instance name.
- 4 Under the directory for the application server you will see directories representing each of the deployed applications in the server. The directories will have the same name as the EAR file that was deployed that contained the application.
- 5 Under each application directory you will find various files and additional sub-directories that contain the application files: JAR files, class files, XML files and webapp files. This is in essence the EAR file "exploded" into the HFS (not exactly, but pretty close). *The files are stored in ASCII, not EBCDIC.*
Note how this means that applications deployed into different J2EE servers are stored in different HFS locations. The example above shows the application PolicyIVP deployed into both APSRV2 and APSRV3. This contributes to application isolation.
- 6 Under the `/controlinfo/envfile` directory you will see a directory with the name of your sysplex.
- 7 Under the sysplex name directory you will find directories with the same name as the server *instances* (not the higher-level server name, but the instance name).
- 8 Finally, under the instance-name directories you will find the various configuration files for the server instance: `current.env`, `webcontainer.conf`, etc.

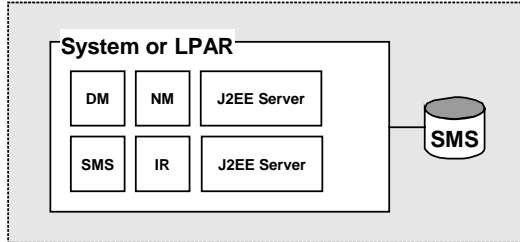
Generally speaking, the SMS modifies and updates these files on your behalf. You won't be climbing into these directories much (and you won't be climbing into the application directories at all). This information was provided to illustrate the way the information is separated based on application name, server name and server instance name.

Note: Copying application files from the WAS HFS to another HFS location is *not* the way to move an application. In addition to the HFS files, information about the application is stored in DB2 and LDAP. Let the SMS do the work under the covers. Don't try to reverse-engineer this because you'll just get yourself into hot water.

WAS "nodes"

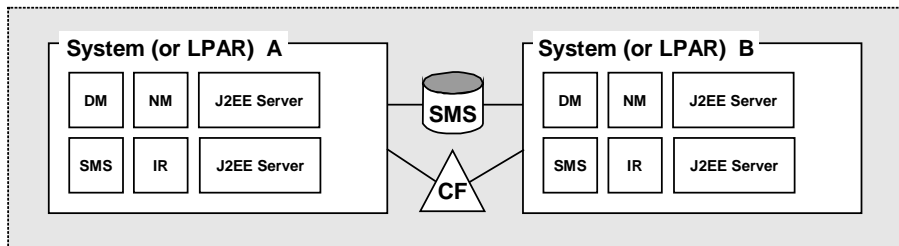
You may think of a WAS "node" as being comprised of all those servers that share a common Systems Management (SMS) repository (which is made up of an HFS "config path" and a set of DB2 tables full of configuration data). This concept is important because you achieve much greater isolation if your test and production are different WAS nodes. If they share the same WAS node, that means they share the same SMS repository, and that means not as much isolation.

A single-system (monoplex) WAS node ...



"SMS" in these pictures is meant to mean the WAS DB2 tables, the configuration HFS and the LDAP information

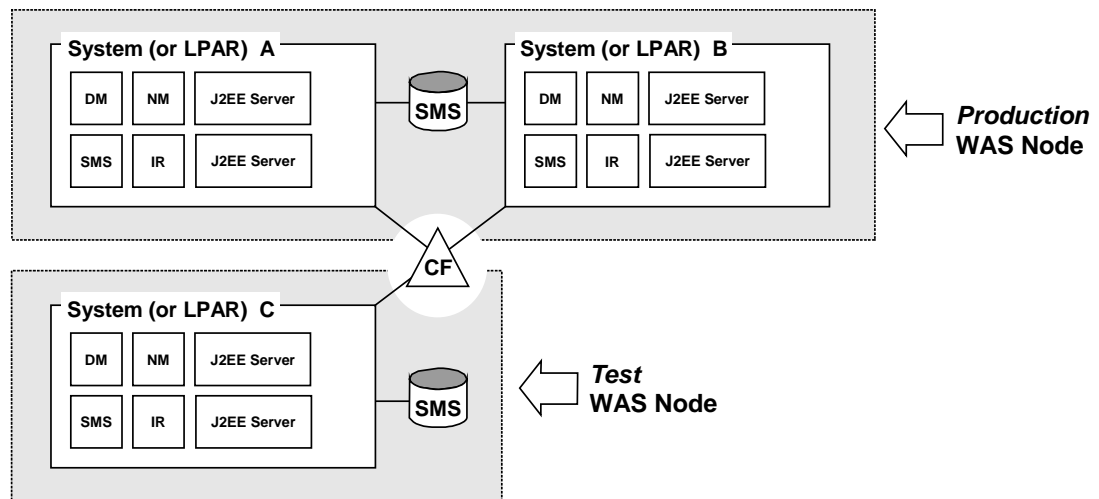
A multi-system (sysplex) WAS node, sharing SMS repository ...



The concept of a WAS "node": a single-system and a multi-system illustration

Question: Is it possible to have more than one WAS node in a sysplex?

Yes. It is possible to have more than one WAS node within a given sysplex:



Possible: multiple WAS nodes in a sysplex

There are three key considerations in accomplishing this:

- The WAS server names used in one WAS node *must be different* from the WAS server names used in the other. Despite the fact that each node would have its own

WebSphere V4.0.1 – Test and Production Environments

configuration HFS and SMS repository, the server names must be different. This is why it's important to have a standardized naming convention going into the configuration of this scenario.

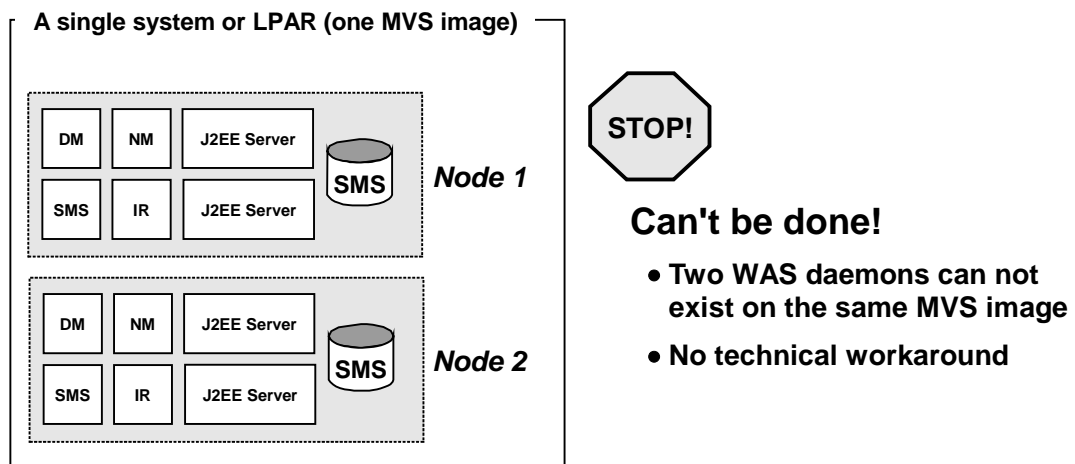
- Only one WAS node in a sysplex may be a multi-system node. Any other WAS nodes configured in the sysplex must be *single-system nodes*. You may make that one multi-system node your production node or a test nodes. For the remainder of this document, it will be assumed that if a multi-system node exists, it will be the production node.

Why? It turns out the Daemon code, if part of a multi-system WebSphere node, issues and listens for ENFs to keep track of the comings and goings of servers in the node. This is a sysplex-wide thing. Two multi-system nodes in a sysplex implies two Daemons issuing and listening for these ENFs. The present design of WebSphere isn't designed to handle this scenario, so the restriction of only one multi-system node is imposed. The new `DATASHARING=0` environment variable tells the WebSphere daemon this: "You are a single-system node. Don't issue or listen for ENFs."

- To allow a WAS node to span systems in a sysplex, the DB2 subsystems that support the shared SMS repository need to be defined as part of a sharing group. In the picture above DB2 subsystem A would be grouped with DB2 subsystem B. DB2 subsystem C isn't sharing with any other system

Question: Is it possible to have more than one WAS node within a system or LPAR?

No. It is *not possible* to have more than one WAS node within a system or LPAR. There is no way to get around this restriction.



Not possible: two WAS nodes on a single system or LPAR

The motivation behind this configuration is to achieve node-level separation without having to carve off a separate LPAR. If you are limited to a single MVS image, then your options are limited to a single WAS node. That limits your options for test and production to maintaining separate WAS J2EE Servers, which is covered under "Conf #1 - Different WAS J2EE Servers on the same system or LPAR" on page 17.

Sharing within a sysplex

What permits a WAS "node" to span more than one system in a sysplex is the ability to share resources across systems. When a WAS node spans systems in a sysplex, the following things are shared:

WebSphere V4.0.1 – Test and Production Environments

- The DB2 tables that make up the WAS configuration and information repository (a bit less than 100 tables make up this repository).
- The WAS configuration and environment HFS, which for many people is at mount point /WebSphere390/WAS401 (or something like that). This is where things like environment variable files, JVM property files, deployed application class files and such are kept.
- The DB2 tables that make up the LDAP naming repository (this is where JNDI information is kept).
- The WLM information (WAS make extensive use of WLM to start up additional server regions as workload requires)
- The RACF database, containing all the userid and other information used by the various WAS components

Because sharing is taking place, you need to be careful when configuring your WAS installation, particularly as that relates to the *naming convention* you employ. Sharing implies the potential for conflict. Sharing also implies something less than total isolation between your test and production environments.

The Systems Management End User Interface (SMS EUI)

The SMS EUI is a graphical workstation tool that provides an interface to the SMS functions of WAS running on zOS or OS/390. It is an essential part of administering your WAS runtime; it is virtually impossible to perform all the necessary tasks without it.

The SMS EUI tool is installed on the workstation, and when started it makes a connection to the SMS server you specify. You log onto the SMS server with a userid (the SMS administration ID) and password. Once logged on, you have control of the SMS and can add, modify or delete definitions maintained by the SMS.

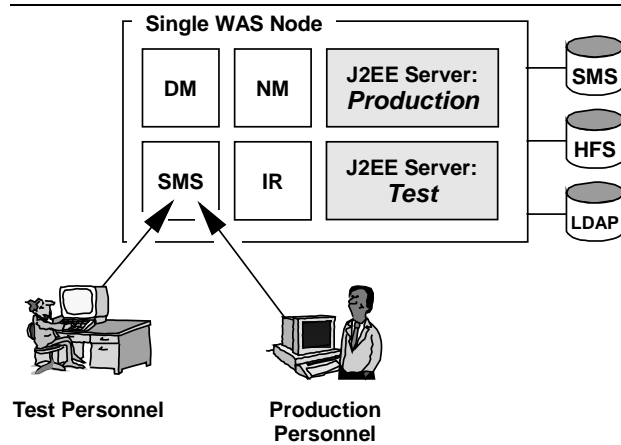
Here's why this topic is being brought to your attention:

- The SMS *will* permit more than one person to log onto the administrative interface simultaneously with the same userid. The SMS *won't* tell you how many people are logged on, and it *won't* keep person "A" from stepping all over person "B's" work. Therefore, you must provide a manual way to coordinate who is on the SMS for a given WAS node. The SMS itself won't manage that.
- The SMS does *not* provide differing access authority based on the userid. It's an "all or nothing" implementation. If you have the password to the SMS administration ID and can log onto the SMS, you have full authority to do good things or do bad things. It is not possible to create a logon ID with just "view" authority, for example.

Given this reality, the question becomes how to minimize the impact of this in a test and production environment. This depends on whether you have a single WAS node for test and production, or multiple WAS nodes.

Single WAS node test/production configuration

A configuration with only one WAS node – and therefore only one SMS – offers little in the way of isolating test personnel from production personnel and the use of the SMS EUI:

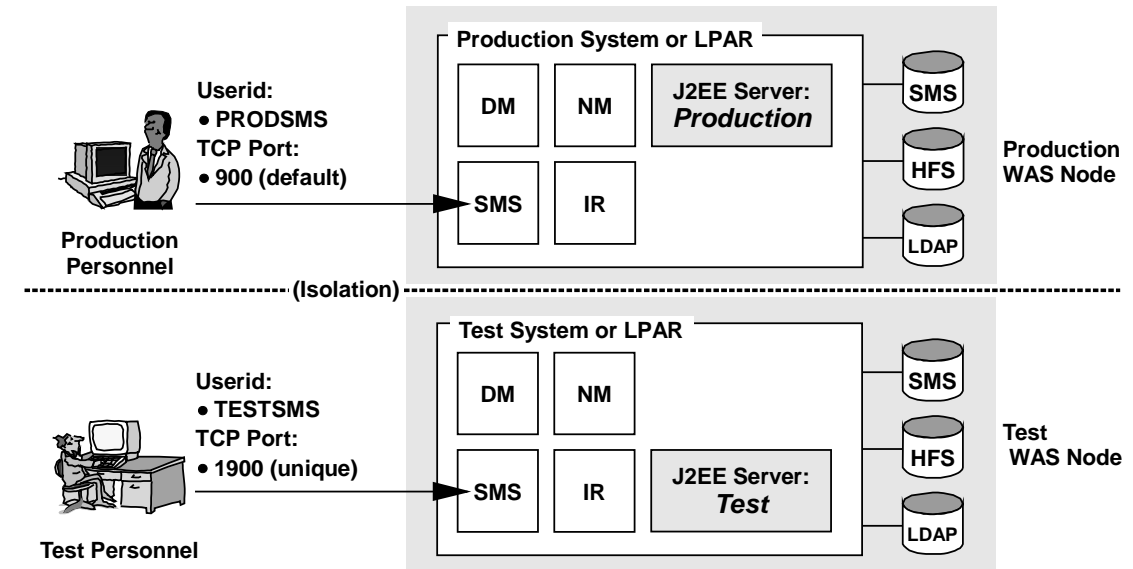


A single WAS node configuration has one SMS; test and production personnel may conflict

Given your choice of a single WAS node configuration for test and production, the issue of potential conflict in SMS EUI usage comes down to human coordination. *You must provide this coordination.* This goes back to the earlier definition of "Human, or Organizational Isolation" (see page 2). Little is offered in a single WAS node configuration.

Multiple WAS node test/production configuration

Configuring multiple WAS nodes (whether in one sysplex or across multiple sysplexes) affords a great deal more isolation of the SMS function:



Multiple WAS nodes provides separation of the SMS and thus greater isolation

As the picture shows, not only does this provide separate SMS servers, but it also gives you the opportunity to create separate SMS administration IDs. You can provide the test people with their ID (TESTSMS in this example) and production with theirs (PRODSMS). Presumably the testers would not know the PRODSMS password, nor would production

WebSphere V4.0.1 – Test and Production Environments

people know the TESTSMS password. Human (or organizational) isolation can be achieved.

The WAS naming convention

A WAS system is comprised of four "base servers," some number of J2EE servers, an LDAP server, various RACF userids, passwords and groups, WLM application environments and JCL procedures. All of these things have names, and you are pretty much free to name them whatever you'd like.

Note: The only restriction is this: the WLM application environment name must be the same as the the server name (not server instance, but the higher-level server name).

However, a structure as complex as WAS calls for some kind of naming convention to keep things straight. This becomes even more important when you have a multi-system WAS node (configuration #2) or a two WAS nodes in a sysplex (configuration #3).

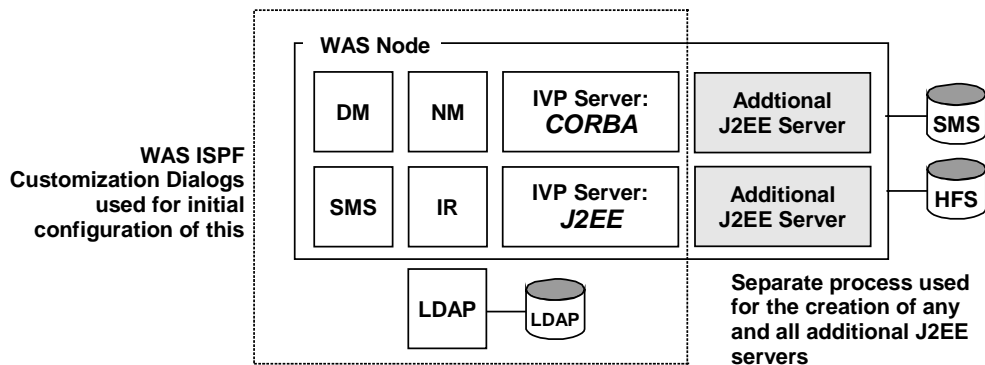
The things most typically worked into the naming convention are:

- Some indication of the resource being related to WAS
- Some indication of the node (test vs. production) to which this resource is related
- An indication of system on which the resource resides if the configuration is a sysplex

There is no such thing as a perfect universally naming convention: each customer will have a slightly different take on this. The point is to come up with a naming convention that makes sense to you. ("Appendix A: Naming Checklist" on page 37 gives you a checklist of the names you'll be asked to provide when defining the "WebSphere Customization" of the ISPF dialogs and when creating J2EE servers).

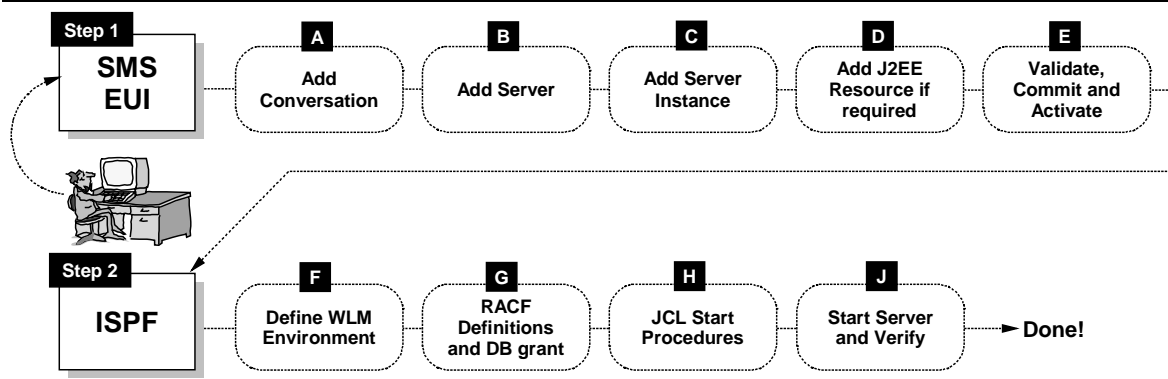
Adding J2EE application servers after initial configuration

The ISPF customization dialogs are designed to create the initial environment, consisting of the the four base servers (Daemon, SMS, Naming and Information Repository), as well as the two IVP application servers (one CORBA server and one J2EE server).



ISPF customization dialogs are not used to create additional J2EE servers

If you wish to create additional J2EE application servers, that is done with a little bit of manual setup on your part. The ISPF panels are *not* used for additional J2EE application servers:



Steps you take to create additional J2EE servers

This document isn't intended to be a step-by-step "how to" manual for this process, so it won't be covered here. The key point is this: if you wish to create additional J2EE application servers, you do not use the ISPF customization dialogs to create them.

Note: You can make use of your initial ISPF dialog work when creating new J2EE servers, however. The BBOWBRAK exec found in the `INSTALL.DATA` file created by the initial running of the ISPF customization panels can be used to generate the RACF commands needed by a new J2EE Server. For instructions on how to do that, go to:

<http://www.ibm.com/support/techdocs/atmastr.nsf/PubAllNum/TD100586>

JNDI names applied to deployed applications

The LDAP repository is used to hold the JNDI paths and names for your deployed EJB beans and webapps. These names are set at the time the application is deployed into the J2EE server.

When one component of your application needs to access another – for example, a servlet client needs to access a session bean, or a session bean needs to access an entity bean – it performs a JNDI lookup of the target object and if found, is returned an instantiated copy of that object.

An application may make reference to these JNDI names either through *symbolic references*, or by having the JNDI names *hard-coded* into the application.

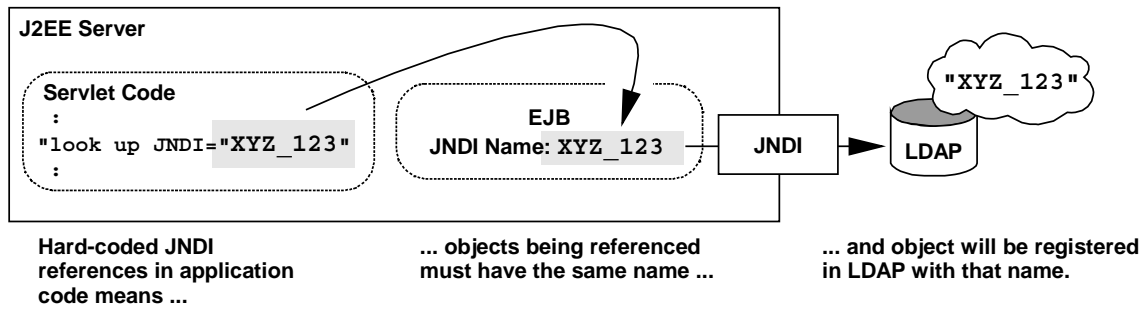
Note: Symbolic references ("java:comp" lookups) became available with the EJB 1.1 specification. The EJB 1.0 specification did not provide this flexibility.

This in turn determines which of the four configurations discussed in the document you may consider:

<i>Type of JNDI reference</i>	<i>Configurations Available to You</i>
Hard-coded JNDI references	Configurations 3 and 4 (<i>not</i> 1 and 2)
"java:comp" symbolic references	Configurations 1, 2, 3 and 4

Your configuration choices may be limited by the JNDI references in your application code

For an application with hard-coded JNDI references to work, the objects being referenced must be deployed with the same hard-coded names:



Hard-coded JNDI references means object must be deployed with the same name

That means the JNDI names for the EJBs will be the same in both your test and production environments. *If you have a single JNDI repository shared between test and production, you will have a naming conflict when you try to register an object when a duplicate name already exists.* Configurations #1 and #2 have a single JNDI repository (LDAP servers) shared between test and production, while configurations #3 and #4 have separate LDAP servers. Hence, *with hard-coded JNDI references you can not use configurations #1 or #2.*

Note: That last statement might be a bit strong. It is possible to use a `.properties` file to provide the JNDI name for an EJB 1.0 application. A `.properties` file must reside in the server's CLASSPATH, and because you want your Test server to be isolated from your Production, it means you would provide *unique* CLASSPATHs for Test and Production. No problem there, but what this does imply is that the `.properties` file for the application *must be modified* when moving the application from test to production (to change the JNDI reference from the "Test" value to the "Production" value). Again, no problem with doing that, but it's one more thing to do and one more thing that might go wrong.

Application code with symbolic ("java:comp" – EJB 1.1 specification) references does not have this restriction. A symbolic reference allows the tooling – the SMS EUI tool – to resolve the symbolic references to JNDI names at the time of deployment. That means the JNDI names of the referenced objects can be pretty much anything. In an environment where test and production share a JNDI repository (configurations #1 and #2), you simply make certain the JNDI names are unique when deploying into one versus the other.

If you use the SMS EUI tool's "default JNDI names" button during deployment, the JNDI names will *not be the same* (which is good). The "default JNDI names" mechanism uses the server name (which will be unique in the sysplex) as part of the JNDI name (actually, the JNDI *path*, but close enough) to insure that the generated JNDI name is unique.

Therefore, if your application code has hard-coded JNDI references, you have a choice:

- Choose either configuration #3 or #4 as illustrated in this document
- Modify your application code and change all hard-coded JNDI references to the more J2EE-compliant "java:comp" symbolic lookups.

Question: what exactly is that long "default JNDI name" string SMS EUI sets?

When you click on the "default JNDI path/name" button in the SMS EUI, it attempts to create a unique JNDI name by making use of six pieces of information it knows about:

```

/WSLPLEX/WASASR2/PolicyIVP/policybmp_deployed/PolicyBMP/com.ibm...
  Sysplex      Server      EAR File      JAR within EAR      Bean Name      Home
                                     Interface
    
```

Structure of the "default JNDI path/name" set by SMS EUI

WebSphere V4.0.1 – Test and Production Environments

So if you deploy an application into separate J2EE servers, you will be guaranteed uniqueness of the JNDI names because the server name will be different.

Application fallback and version control

Imagine this scenario: you have an application running in production. (Which configuration you choose doesn't matter for this discussion: this topic is generic to them all.) You make a modification to the application and test it in your test environment and everything looks good, so you roll it into production. Over in production you spot a problem you missed in test. How do you fall back to your most recent verified version?

Bottom line: you redeploy the application into the WAS J2EE server. There is no way within the WAS 4.0.1 runtime to select a previously installed version of an application and simply fall back to it.

That brings up this key point: your version control mechanism is best maintained in the development environment, not in the WAS 4.0.1 runtime. If the need arises to fall back to a "known good" version of an application, you get the application from your development system and then go through the steps required to redeploy the application into your runtime.

Multiple Environment Configurations

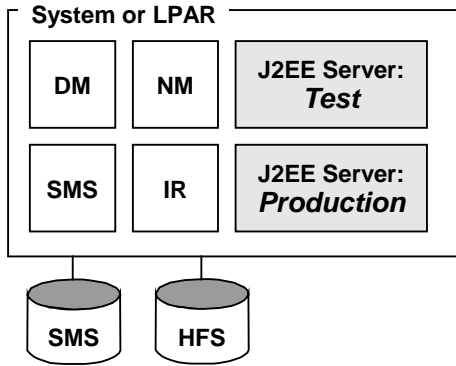
There are four basic configurations you can use to achieve some degree of isolation between test and production environments. There are, of course, variations on each of these. This document will not try to cover all combinations, but rather offer the key considerations that you should keep in mind.

Overview of the four configurations

<i>Configuration</i>	<i>Description</i>	<i>Page</i>
Conf #1 - Different WAS J2EE Servers on the same system or LPAR	This configuration can be accomplished on a single system or LPAR, and is done within a single WAS "node." It is quickest and easiest, but offers the least isolation.	17
Conf #2 - Different WAS J2EE servers on different systems in the same WAS node	This configuration is similar to the first, but the test and production application servers are on different systems or LPARs in the WAS node.	22
Conf #3 - Different WAS nodes within the same Sysplex	This configuration involves configuring a separate WAS node within a Sysplex. It requires care to make sure you don't use duplicate resource names. Once done, it offers a fairly good level of isolation.	25
Conf #4 - Different WAS nodes in separate sysplexes	This is the ideal configuration if complete isolation is the goal. It requires the most resource.	35

Each configuration is discussed at starting at the page number provided.

Conf #1 - Different WAS J2EE Servers on the same system or LPAR



Test environment in one WAS application server, Production in another, both on same system

Snapshot rating of configuration

<i>Attribute</i>	<i>Rating (more stars = "better")</i>
Ease of configuration	★ ★ ★ ★ ★
Ability to minimize resource usage	★ ★ ★ ★ ★
Degree of <i>application</i> programming isolation	★
Degree of <i>system</i> programming isolation	(none)
Degree of human isolation	★

Design Assumptions

- The WAS runtime is configured as a monoplex and the WAS node is run entirely within one system
- You have some number of "test" J2EE servers and some number of "production" J2EE servers.
- You have implemented some form of organizational procedure to control access to the SMS administrative function.

Detail on "Ease of Configuration"

This is the easiest to configure because there's no WAS base server configuration beyond the initial configuration. Because you have but one system on which you plan to run WAS, you have only one node and only one set of base servers. One running of the ISPF customization dialogs is all that is necessary. The creation of the J2EE servers is a relatively easy effort common to all the configurations described in this document.

Detail on " Ability to minimize resource usage"

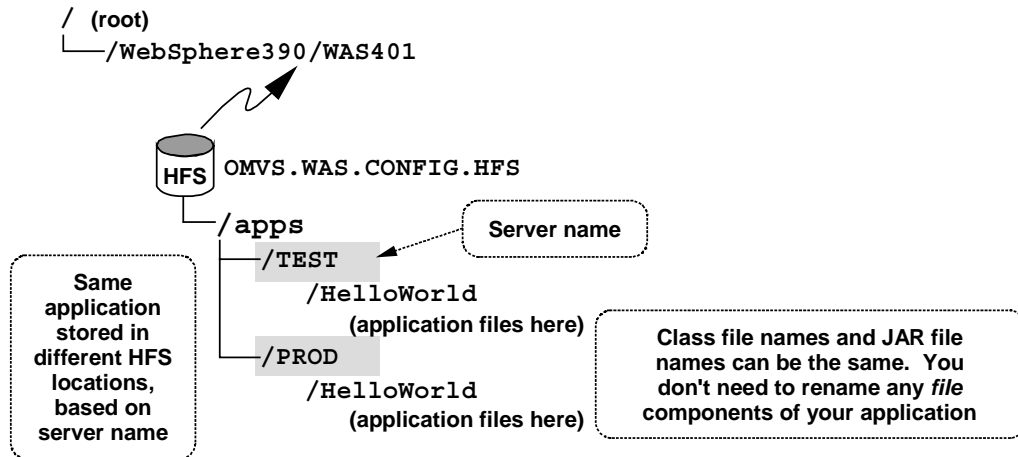
This configuration has the highest rating for this category simply because it uses the minimum amount of resources. This configuration is recommended only if you do not have the resources to create a configuration #3 or #4 as illustrated in this document. The degree of isolation is so low that one should consider this configuration only as a last resort.

Detail on "Degree of *application* programming isolation"

There is actually more application isolation than you may initially think. When an application is deployed into a WAS runtime, the application code – or more precisely, the

WebSphere V4.0.1 – Test and Production Environments

contents of the EAR file you deploy – are placed into the HFS in a directory structure isolated by J2EE server name and further isolated by the name of the application:



Applications deployed in different servers end up in different locations in the HFS

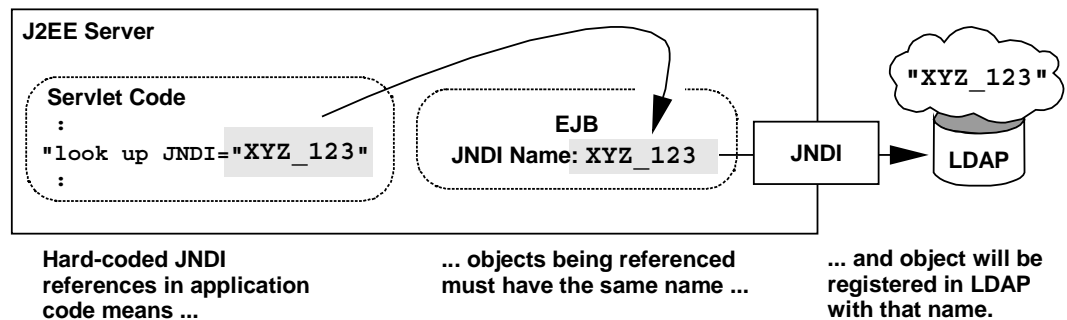
Since the design of this configuration calls for "test" and "production" to be isolated by using different J2EE servers – which *must* have different server names -- you know that an application deployed into your test server will be placed in the HFS down a different branch of the tree structure than one deployed into the production server.

This configuration does have some limitations, however. They are covered next.

JNDI Names

With your test and production environments on the same system and in the same WAS node, your applications then by definition share the same local JNDI namespace. JNDI (implemented with LDAP on S/390) requires that all registered components have unique names. That means your test application must have *different* JNDI names from the same application deployed into production. *That by itself is a not the primary limitation.* The limitation comes in when your application uses hard-coded, explicit JNDI references rather than "java:comp" symbolic lookups.

The following picture illustrates why this is a limitation:

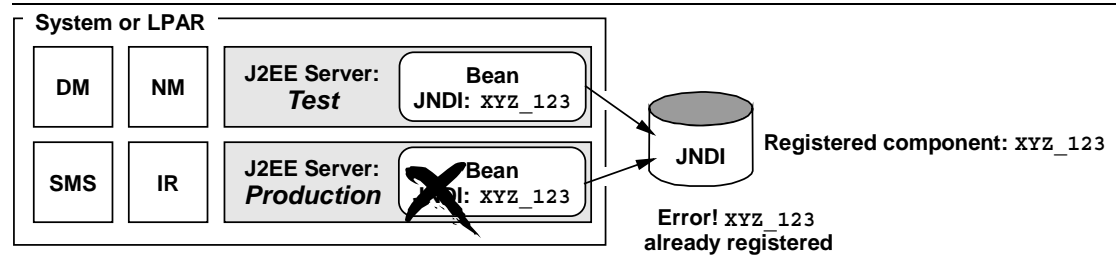


Hard-coded JNDI references means object must be deployed with the same name

Therefore, in this configuration, an application with hard-coded JNDI references can be deployed in your test server, or your production server, *but not both at the same time.* If you deployed an application into your *test* server with a JNDI name of XYZ_123, and then attempted to deploy the same application into your production

WebSphere V4.0.1 – Test and Production Environments

server with a JNDI name of XYZ_123, the deployment into production would fail: JNDI would report a duplicate resource name:



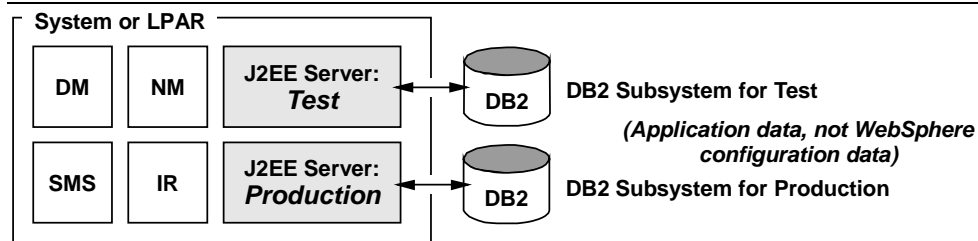
Deployment failure because of JNDI name already registered

This is a limitation of any configuration where a shared local JNDI namespace exists (configurations #1 and #2). Separate WAS nodes (configurations #3 or #4) avoids this limitation.

Note: You can eliminate this restriction by using "java:comp" lookups in your application code. The recommendation is to use them whenever possible and to avoid explicit JNDI references.

J2EE Resource Names

With your test and production environments on the same system and in the same WAS node, you may wish to define separate data subsystems (DB2, CICS, etc.) for test and production *application data*:



Separate data subsystems for test and production (DB2 used as an example)

Note: The preceding picture is simply an illustration. There is no requirement that says your Test and Production application data must be housed in different data subsystems. It is illustrated that way simply to drive home the key point, which is this: to WebSphere, different data subsystems imply different J2EE data resource names.

Using a single data subsystem for both Test and Production application data would imply the use of a single J2EE data resource for both. It would then be up to you to make certain that applications under test make use of the test data, and not accidentally go against the production data.

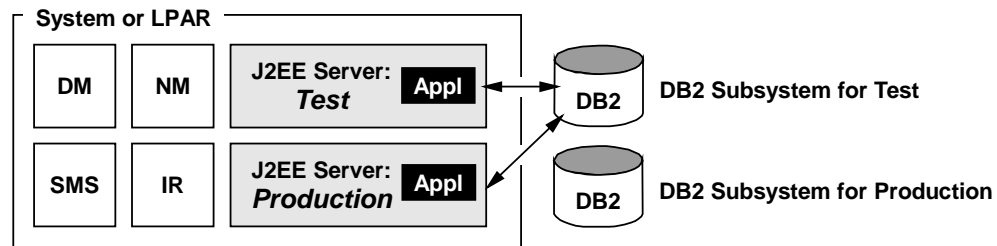
In the J2EE application world, underlying data resources are associated with *J2EE Resources* (and resource *instances*), which are defined in the SMS administrative tool. At deployment time you tie an application's data reference to one of the defined J2EE Resources, and when that application calls for data, the request is mapped to the right DB2 or CICS subsystem, based on the J2EE Resource definition.

For this single-system configuration, this becomes important for this reason: J2EE resources are defined at the *sysplex level* and are available to all the J2EE application servers you have in your WAS node. A test DB2 subsystem, for

WebSphere V4.0.1 – Test and Production Environments

example, will be defined with one J2EE resource name (TEST_DB2) and the production DB2 subsystem would be a *different resource name* (PROD_DB2).

What that does is introduce a slight difference in the *process* required to deploy the application in test vs. production: the deployer must know to resolve the proper data resource. Therefore, the *possibility* exists that an application deployed into production might be accidentally pointed to the test data subsystem, or worse, a test application pointed to production data:



Different J2EE Resource names means deployment process can't be identical

Contrast this with "Adding J2EE application servers after initial configuration" illustrated on page 33. Because in configuration #3 (and configuration #4 as well) two different WAS nodes are in play, you can define the same J2EE resource name for both the test and production environments. With the J2EE resource names the same, the process of deploying the application is identical when it comes to resolving the resource.

Web application virtual host and context root settings

Web applications are uniquely identified by the WAS system using the concatenation of two values: the *virtual host* and the *context root*. The virtual host is the IP host name used to route the browser request to the proper server, and the context root is used to resolve the request down to one of what may be many webapps deployed in the server.

Note: This topic is covered in far greater depth in the white paper WP100238, which can be downloaded from the web at <http://www.ibm.com/support/techdocs>. Go to the "White Papers" link and search on the number WP100238.

WAS will not be able to properly distinguish a test webapp from a production webapp if they are deployed using the same virtual host and context root values. This holds true even if the the two webapps are deployed in different J2EE Servers. WAS will not flag this at the time of deployment.

Therefore, you should plan on having a different "virtual host/context root" value for your test webapps than used for your production webapps.

The easier of the two to change is the virtual host value used. The context root setting is made in the `application.xml` deployment descriptor inside the EAR file, and is set by the AAT at the time you assemble the application. The virtual host value is defined in the J2EE server's `webcontainer.conf` file. Changing the virtual host rather than the context root allows you to minimize the changes to the *application* between test and production.

WebSphere V4.0.1 – Test and Production Environments

Therefore, you should plan on using a different virtual host value in your test J2EE server than is used in your production J2EE server. Requests issued from a browser intended for your test environment will contain the IP host name and port of your test environment's virtual host setting, and will drive the webapp in the test environment only. Your production J2EE server would have a different virtual host, and a URL intended for the test environment would *not* be able to invoke a production webapp.

Changing something as simple as the port number effectively changes the virtual host value. So `www.mycompany.com` is a different virtual host value from `www.mycompany.com:8080`.

Though *critical* in configurations #1 and #2, you should follow the general rule of different virtual hosts for all configurations presented in this document. Through some fancy networking techniques you can probably get around this in Configurations #3 and #4, using different virtual host values helps maximize the isolation of test and production, and that's the objective.

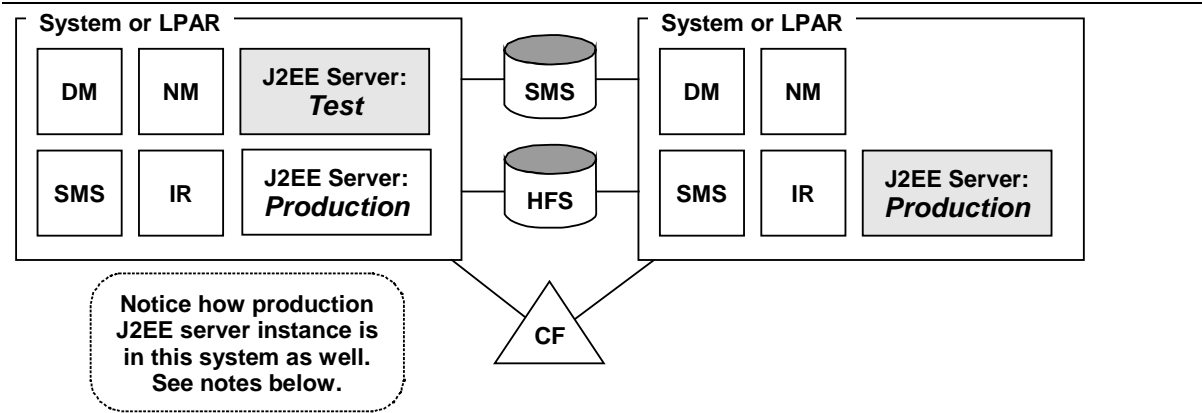
Detail on "Degree of *system* programming isolation"

This configuration has only one zOS system, which means there is only one copy of WAS and only one copy of DB2 on which WAS relies. There is virtually no opportunity for system programming isolation because of the reality that only one WAS node can exist on an MVS image. Hence a rating of "none".

Detail on "Degree of human isolation"

This configuration has a very low rating in this category because of the limited isolation provided by the SMS administrative function (see "The Systems Management End User Interface (SMS EUI)" on page 12 for a description of the limitations inherent in the product). There is considerable exposure to your production environment if different people are using the SMS EUI to access the system. This exposure is magnified in an organization where the test and production personnel are physically or organizationally separated and close coordination doesn't occur. If the environment is relatively small and one person – or a close team of a few people – has access to the SMS administrative function, then you may be able to limit the exposure *through procedures you implement between your team*.

Conf #2 - Different WAS J2EE servers on different systems in the same WAS node



Test and Production in different application server, each on a separate system within the WAS node

Snapshot rating of configuration

Attribute	Rating (more stars = "better")
Ease of configuration	★ ★ ★ ★
Ability to minimize resource usage	★ ★ ★ ★
Degree of <i>application</i> programming isolation	★
Degree of <i>system</i> programming isolation	★ ★
Degree of human isolation	★

Design Assumptions

- Two or more systems are configured in a sysplex
- Only one WAS node has been configured, and it spans the systems in sysplex
- Test and Production is maintained as separate J2EE servers
- You have, or plan to have, *production* J2EE server instances on both systems in the sysplex. If you have no plans to do that, and you intend on separating test and production as separate J2EE servers on different systems, then you are better off going with Configuration #4 and configuring each system as a monoplex.

Note: This configuration is an extension of Configuration #1. The reason this configuration is included is it provides a degree more system programming isolation because two or more physical systems (or LPARs) are present. Some flexibility is offered for applying fixes to one system and not the other and "rolling" the changes as described in Chapter 6 of the "Installation and Customization" guide (GA22-7834). *Therefore, this configuration provides the opportunity to apply updates to WAS an not disrupt service to clients.* That topic is a big one, and best left covered in the "Installation and Customization" guide.

The degree of *application* separation and *human* separation is largely the same as in Configuration #1 because of the way WAS spans a sysplex.

Detail on "Ease of Configuration"

This configuration is less than Configuration #1 because configuring a multi-system WAS node involves creating unique server *instances* (not the server, but the instances)

WebSphere V4.0.1 – Test and Production Environments

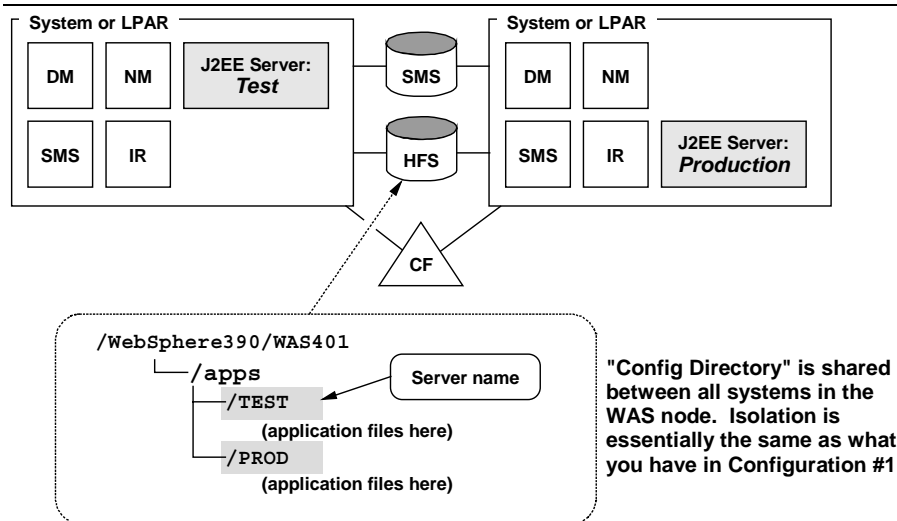
on each system. The server name itself is a higher-level thing and spans the systems in the WAS node. If your naming convention is constructed adequately, the server instance names will have an indicator of the system on which the instance lives, and the act of configuring the instances should be a relatively small thing. (See "The WAS naming convention" on page 12.)

Detail on " Ability to minimize resource usage"

This configuration entails *at least* two systems or LPARs, otherwise it becomes Configuration #1. The extent to which you expand the use of resources beyond two systems is up to you.

Detail on "Degree of *application* programming isolation"

Due to the shared nature of WAS across systems in a multi-system WAS node, this configuration really doesn't offer any greater isolation of your application than does Configuration #1. *This holds true even when your test J2EE server instance is on one system and your production J2EE server instance is on another.*



Configuration Directory is shared; application isolation the same as in Configuration #1

The same considerations offered under Configuration #1's "Detail on "Degree of *application* programming isolation"" on page 17 apply here. That would include the issues of hard-coded JNDI references, deployment differences involving different J2EE resource names and the issue of webapp virtual hosts and context roots.

Detail on "Degree of *system* programming isolation"

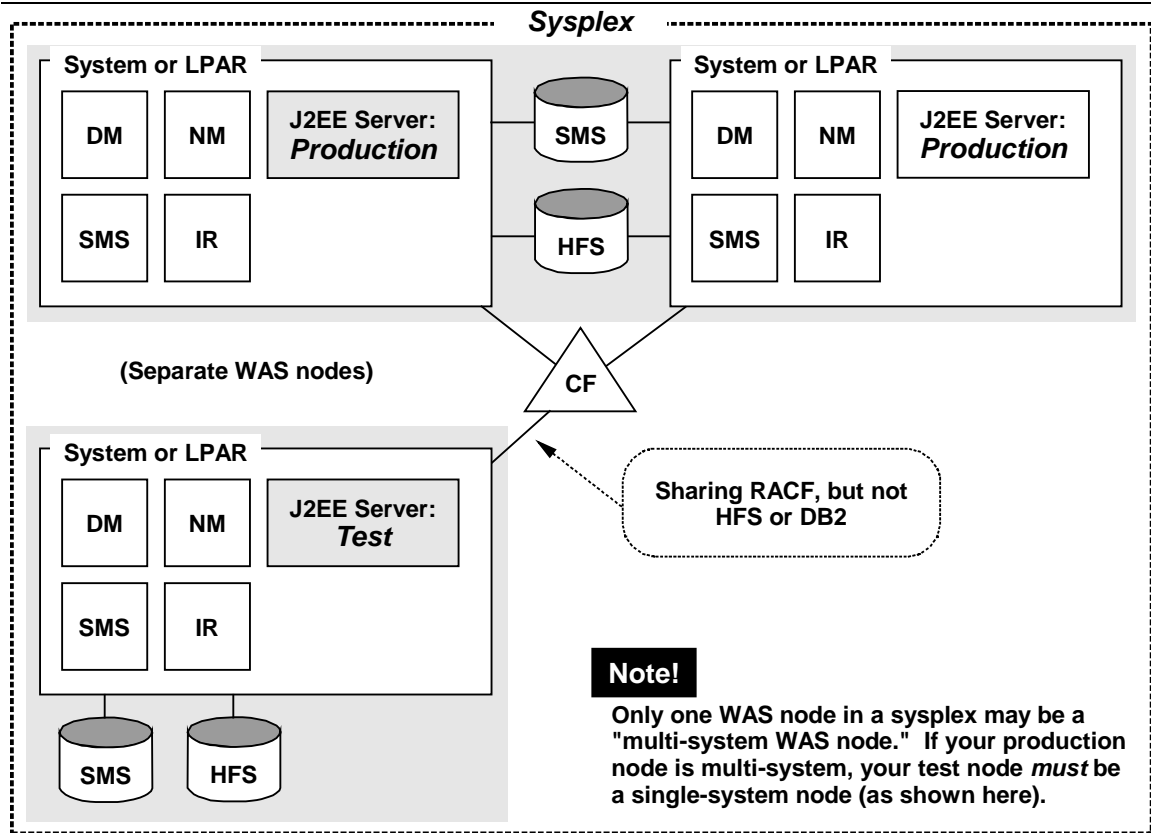
Here's where this configuration offers value over Configuration #1. Because at least two systems are in play, it is possible to configure the HFS so that each system has an *unshared* place where the WAS code is installed. This allows you to apply fixes to one system and restart that system while keeping the other system up and running. This is known as a "rolling upgrade."

The topic of rolling upgrades and the various methods by which you can start your systems ("cold," "warm," "hot" and "quick") are detailed in Chapter 6 of the "Installation and Customization" guide (GA22-7834). The key point here is because you have separate systems (or LPARs), additional flexibility is available to you for system programming isolation.

Detail on "Degree of human isolation"

No better than that offered under Configuration #1. The SMS administrative interface is shared across all systems in a WAS node. The same holds true for the system console. If you have not yet reviewed the section titled "The Systems Management End User Interface (SMS EUI)" on page 10, it would be beneficial to do so at this time.

Conf #3 - Different WAS nodes within the same Sysplex



Test on separate WAS node in Sysplex and not sharing with Production WAS node

Snapshot rating of configuration

Attribute	Rating (more stars = "better")
Ease of configuration	★
Ability to minimize resource usage	★★★
Degree of <i>application</i> programming isolation	★★★★★
Degree of <i>system</i> programming isolation	★★★★★
Degree of human isolation	★★★

Design Assumptions

- The system (or LPAR) on which the test WAS node is configured is dedicated to application and software test. It has dedicated non-shared HFS.
- The test system is a single-system WAS node.
- The test system DB2 subsystem is *not* part of any DB2 shared group.
- The RACF database is shared between all the systems in the sysplex.
- The scope of enqueues have *not* been changed.

Detail on "Ease of Configuration"

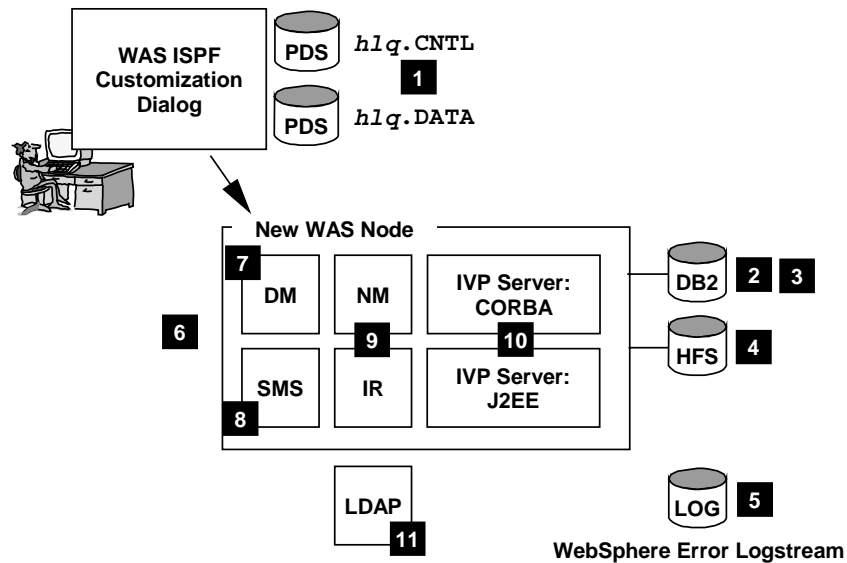
This configuration has the lowest "ease of configuration" rating of the four configurations. This is because configuring two WAS nodes in the same sysplex requires the following:

- The server names in the test node (base servers and J2EE servers) *must* be different from the server names in the production node. The RACF IDs underlying the servers typically have a naming convention tied to the server names, so they too would be different. Planning and care must be exercised to avoid naming conflicts.
- The DB2 subsystems in the test node must *not* be part of the sharing group with the production DB2 subsystems.

Note: There is a key difference in configuring a test WAS node in a sysplex, and it has to do with an environment variable called `DATASHARING`. This is discussed under "Datasharing environment variable and the bootstrap" on page 29.

Understand the things you'll code differently in the ISPF customization dialogs

As you prepare to run through the ISPF customization dialogs to establish the second WAS node within the sysplex, bear in mind the need to have the following variables set with different values from other WAS nodes in existence:



When configuring a second (test) WAS node in the same sysplex, new values are needed

The numbered blockes are described next:

	What	Where specified in dialogs
1	WAS customization data sets – allocate different from those used for production node.	Option 1, "Allocate Target Data Sets"
2	WAS DB2 subsystem – provide information about separate DB2 subsystem that will be used by the test WAS node	Option 2, "Define Variables" Option 1, "System Locations" Panels 1 of 2 and 2 of 2
3	WAS DB2 VCAT value and data and index volumes – provide different values	Option 2, "Define Variables" Option 2, "WebSphere Customization" Panels 1 of 4
4	WAS HFS configuration mountpoint –	Option 2, "Define Variables"

WebSphere V4.0.1 – Test and Production Environments

	What	Where specified in dialogs
	make it different from the mount point used by the production WAS node	Option 2, "WebSphere Customization" Panel 1 of 4
5	WAS Error Logstream – create a DASD-only error log with a unique name for the test node.	Option 2, "Define Variables" Option 2, "WebSphere Customization" Panel 2 of 4
6	WAS common group IDs and unauthenticated userids – while these could technically be the same as on the production node (shared RACF, remember), it is best to have them uniquely identified with your test node.	Option 2, "Define Variables" Option 2, "WebSphere Customization" Panel 3 of 4
7	WAS Daemon base server – provide a unique name for the server along with the associated RACF IDs	Option 2, "Define Variables" Option 3, "Server Customization" Panel 1 of 4
8	WAS Systems Management base server – provide a unique name for the server along with the associated RACF IDs	Option 2, "Define Variables" Option 3, "Server Customization" Panel 2 of 4
9	WAS Interface Repository and WAS Naming base servers – provide unique names and RACF IDs	Option 2, "Define Variables" Option 3, "Server Customization" Panel 3 of 4 (IR) and 4 of 4 (Naming)
10	IVP Servers – if you plan on using them you need to provide unique names and RACF IDs	Option 2, "Define Variables" Option 4, "IVP Customization" Panel 1 of 2 (CORBA) and 2 of 2 (J2EE)
11	LDAP Server – provide a unique server for the test node	Option 2, "Define Variables" Option 5, "LDAP Customization" Panel 1 of 1

This implies that pretty much everything will need to be uniquely named and coordinated. That leads to the next topic, which involves the *planning* for all this.

Planning your server names

One of the key restrictions when configuring more than one WAS node within a sysplex is that all server names must be unique. It doesn't matter that the two WAS nodes are on separate DB2 subsystems. *The names must be unique.*

If by accident you name a server the same name as exists in the production node, the ISPF customization dialogs will *not* flag that as an error. What you will see at server startup is the BBOU0758W message indicating a duplicate server name.

"The WAS naming convention" on page 12 provided background on the issue of a WAS naming convention. The essential points are these:

- Understand the relationship between the different servers and other resources that require a name
- Have a naming convention in mind before you start the WAS ISPF customization dialogs.

WebSphere V4.0.1 – Test and Production Environments

SMS Administration ID

There is no technical reason why the SMS administration RACF ID for the SMS on test must be different from that on production, but *it is strongly recommended they be different IDs*. The SMS ID is set in the ISPF customization dialogs under "Define Variables," "Option 2, WebSphere Customization," "Panel 3 of 4":

```
-----      WebSphere for z/OS
Customization
Option ==>

WebSphere Customization (3 of 4)

WebSphere Common Groups and User IDs

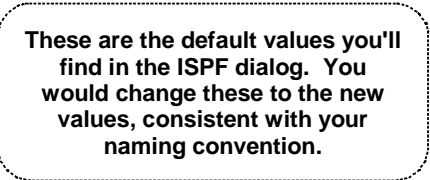
Control region group for ALL servers..: CBCTL1
Control region GID for ALL servers....: 2211
Server region group for base servers..: CBSR1
Server region GID for base servers....: 2201

Unauthenticated User Definitions:
Userid...: CBGUEST      UID...:
Group....: CBCLGP      GID...:

WebSphere Application Installer

Group....: CBCFG1      GID...: 2300

WebSphere Administrator Information
Userid...: CBADMIN      UID...: 2103
Password.: CBADMIN
Group....: CBADMGP     GID...: 2203
```

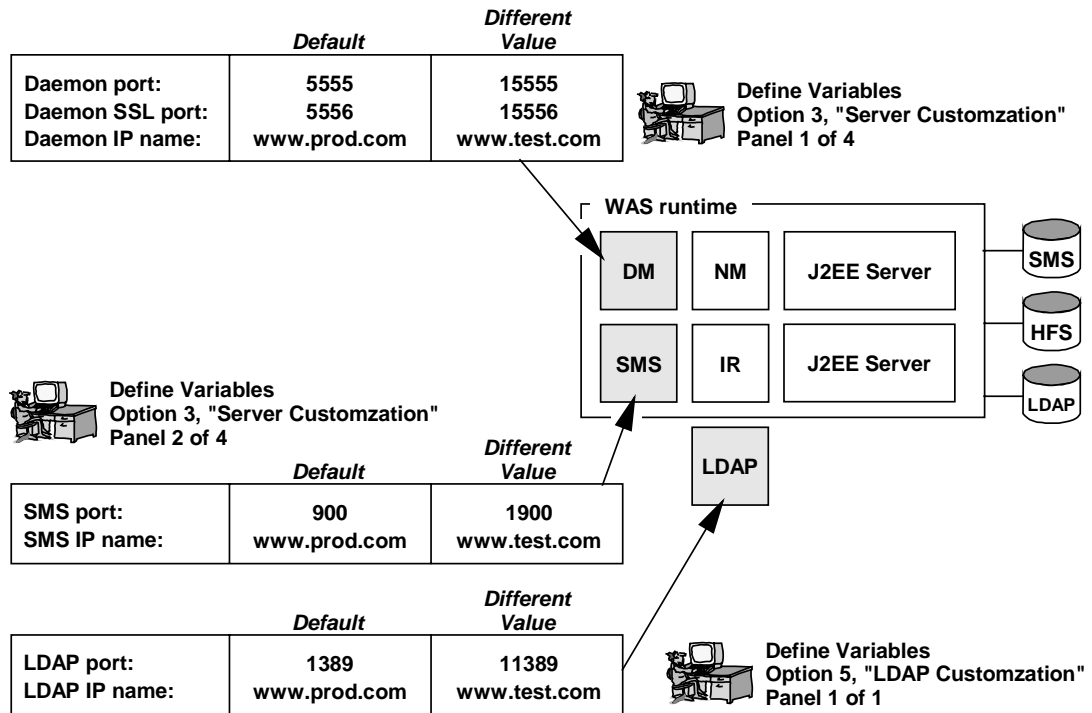


Specification of the SMS administration ID within the ISPF dialogs

Using different SMS administration IDs for test and production allows you to provide human (or organizational) isolation. The test group will be given access to the test node's SMS EUI with the userid/password unique to that node; the production people will have access to the production node's SMS EUI with the userid/password unique to that node. This concept is illustrated under "Detail on "Degree of human isolation"" on page 17.

TCP Ports and IP names

To maximize the isolation between your test and production nodes, you should insure the TCP/IP ports and IP names used for in your test node are different from those in the production node. The following picture illustrates the ports and IP names that should be different:



Use different TCP port and IP names; where in ISPF these are set

Using unique ports creates a stronger differentiation between your test and production environments. The recommendation is to create a unique IP name and unique ports for your servers in the test and

Note: Using common IP names and port numbers between test and production isn't a hard technical requirement (no abends will occur if you have common values). But it is a *strong recommendation* that you provide unique IP names and port numbers for your test servers to differentiate them from production.

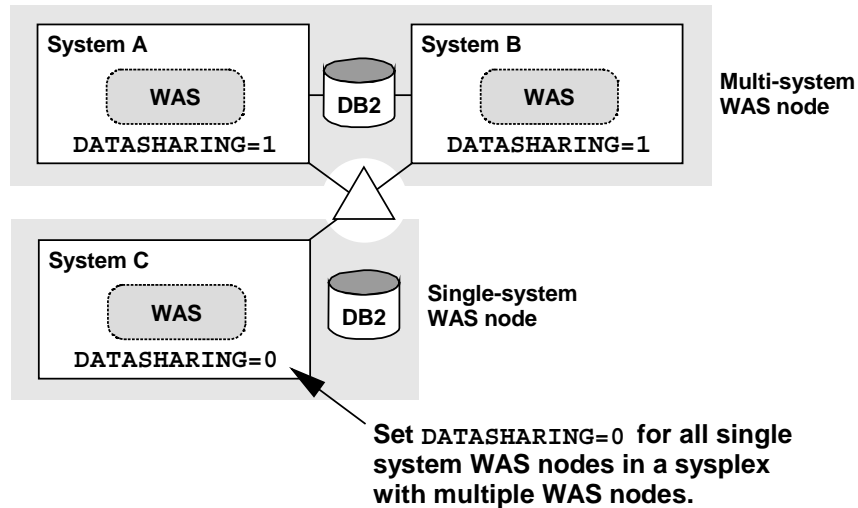
Datasharing environment variable and the bootstrap

A new environment variable called `DATASHARING` was introduced with PTF UQ99328 (Service Level W401014). This variable is used whenever a sysplex has more than one WAS node configured.

One of the key restrictions to the "multiple WAS nodes in a sysplex" configuration is that only one WAS node in the sysplex can be a multi-system node (see "WAS nodes" on page 8 for an illustration of this). Multi-system WAS nodes share DB2 resources between the systems in the node.

Note: This also implies that different DB2 subsystems be used for the production node and the test node. This restriction is further imposed due to the way WebSphere refers to its system tables using a hard-coded owner qualifier of BBO. If you tried to create two node's database tables in one DB2 subsystem, the hard-coded qualifier of BBO would conflict when the second node's tables were created. You can't rename them ... the references to "BBO" is hard-coded in the WebSphere program code.

When multiple WAS nodes are configured in a single sysplex, those WAS nodes that are made up of single systems don't share DB2 resources with other systems. The `DATASHARING` environment variable is used to indicate this:



Single-system WAS nodes set DATASHARING=0; multi-system nodes to 1

The variable is set in the `configuration.env` file just prior the executing the bootstrap phase. The default is `DATASHARING=1`, so if you are setting up a test node in a sysplex and your node isn't sharing DB2 resources, you must code this and provide a value of `DATASHARING=0`.

Configuration using the ISPF dialogs

With a plan for the different names and ports for the new WAS node firmly in hand, you're ready to run through the ISPF customization dialog panels. Before you start, a quick memory checklist:

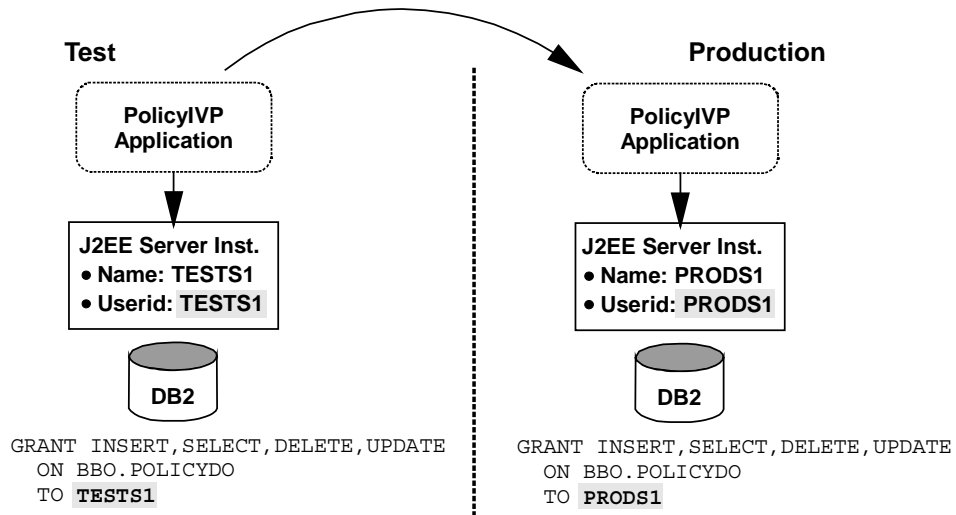
- Separate system or LPAR established? Yes No
- DB2 subsystem not part of any sharing group? Yes No
- HFS separate and non-shared? Yes No
- All other WAS 4.0.1 system preparation tasks performed, as indicated in Chapter 2 of "Installation and Customization" (GA22-7834-02)? Yes No

Run through the panels, create the customized jobs and the submit the jobs in the sequence specified in the `BBOINSTR` member of the `hlq.CNTL` datase, making sure to include `DATASHARING=0` in `configuration.env` prior to running bootstrap.

Authorization to backend data stores

The J2EE application server name on the test system must be different from the corresponding J2EE application server on the production system. That is one of the restrictions of this configuration, but not a restriction in when you have two completely separate sysplexes.

Therefore, if the authorization granted to provide access to backend systems (such as DB2) is based on the server's instance RACF userid, then you must insure that the authorization has been properly granted on both test and production based on the appropriate server instance ID:



Example: DB2 authorization grants on test vs. production when server names different

The picture above illustrates the GRANT used to grant authority to the PolicyIVP table BBO.POLICYDO.

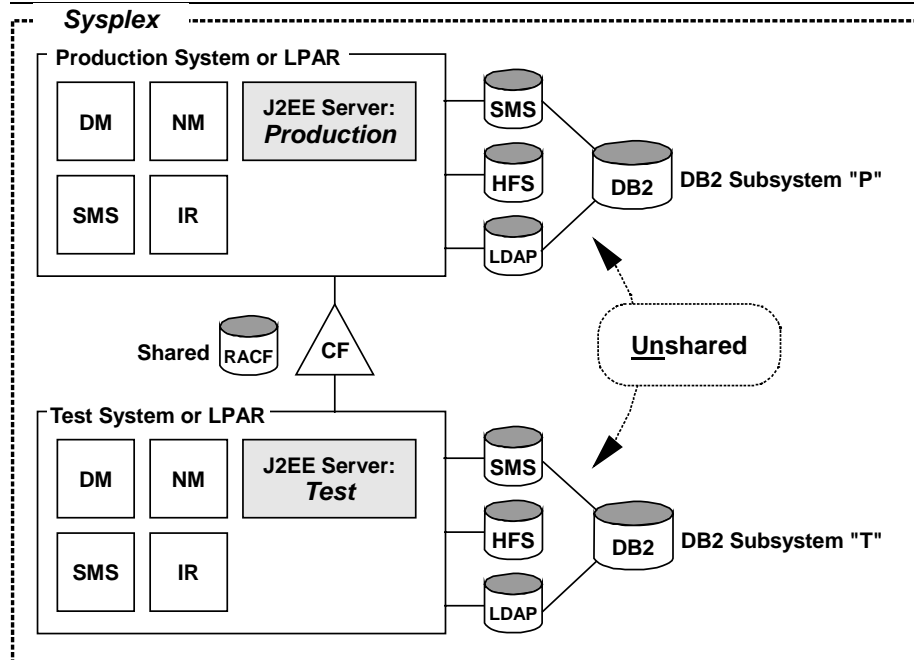
If your application makes use of stateful session beans, you need to grant authorization to BBO.STATEFUL_BEANS as well.

Detail on "Ability to minimize resource usage"

Once you make the decision to deploy a second WAS node within your sysplex, it automatically implies a separate system or LPAR. There is no way to avoid this if a second WAS node is your objective. Therefore, this configuration uses more resource than does any configuration with only one WAS node. The benefit you receive is greater isolation.

Detail on "Degree of *application programming isolation*"

An application deployed in the test WAS node is completely and entirely isolated from applications deployed in the production WAS node:



Isolation of applications in test environment vs. production environment

The picture illustrates the following regarding the test and production application environments:

- They operate in different J2EE application servers operating on different systems within the sysplex. The different application servers can be stopped and started independent from one another.
- They operate in different WAS nodes, managed by different WAS base servers, which run on different systems within the sysplex. The WAS daemons themselves, which controls the whole node, can be stopped and started independently from one another.
- The information WAS maintains about a deployed test application is kept in separate DB2 tables in separate, non-shared DB2 subsystems from those maintained for deployed production applications.

Note: This is not just a benefit, it is a requirement that the two node's DB2 tables be in separate subsystems. The WebSphere code refers to the tables with a hard-coded qualifier value of "BBO." Therefore, the tables must be created with BBO as the owning qualifier. You can't simply rename them.

- The HFS and the HFS directory structure in which the test application (JAR files, XML files, WAR files, etc) is stored is separate and non-shared from the HFS used for the production application. There's absolutely no connection between the two.

JNDI names

Because each WAS node has its own LDAP server, you may deploy an application into your test and production nodes with the same JNDI names and not have any naming conflicts.

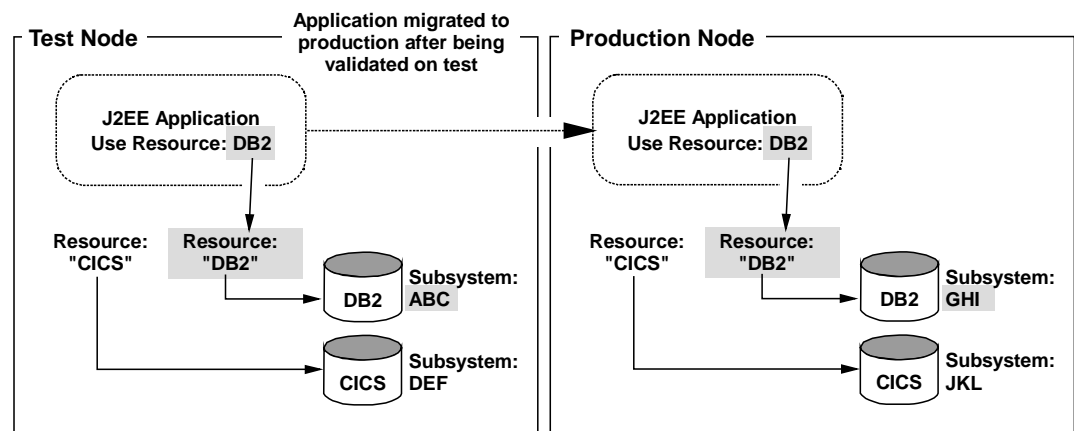
WebSphere V4.0.1 – Test and Production Environments

If your code uses "java:comp" symbolic lookups, you can use the "default JNDI names" button of the SMS EUI and it will generate unique names based on the server name. The names may be different, but the *process* by which you deployed the application into test versus production is the same (you clicked on the "default JNDI names" button).

Data resource names defined to WAS in each node

Within the WAS runtime are definitions known as "J2EE Resources" and "Resource Instances." These are symbolic names that are tied to an actual data source such as a DB2 subsystem or a CICS region. When you deploy a J2EE application that accesses those data sources, you point the application at the resource name, not the actual subsystem name.

To provide as much deployment process transparency as possible, you should keep the "J2EE Resources" names the same on both your test and production systems. The actual data subsystem under the resource on your test system will be different from the subsystem under your production resource, but the symbolic name used by the application will be the same. That piece of the deployment process will be identical:



Keep J2EE Resource names the same; underlying subsystems are different

Web application virtual host and context root values

In this configuration you intentionally configure a different Daemon and SMS host name to insure separation of your test and production environments. A different host name for your test environment means by definition your test J2EE servers will employ a different "virtual host" value from your production environment. This is good, for it helps to increase the separation of the two environments.

Your application context root value may stay the same between test and production.

Detail on "Degree of system programming isolation"

Since the test WAS node is within a separate system or LPAR, and the assumption was that system or LPAR was dedicated to test, you can maintain separation of the operating system and subsystem code. Things like parm libraries, link lists and proc libraries would be isolated from your production system.

Message: the same methods you employ today for test and production within a sysplex apply here. In fact, the WAS product itself can be viewed just like any other subsystem in terms of isolation for testing purposes.

What is shared in this configuration

The two WAS nodes still exist within a sysplex, and do share some resources:

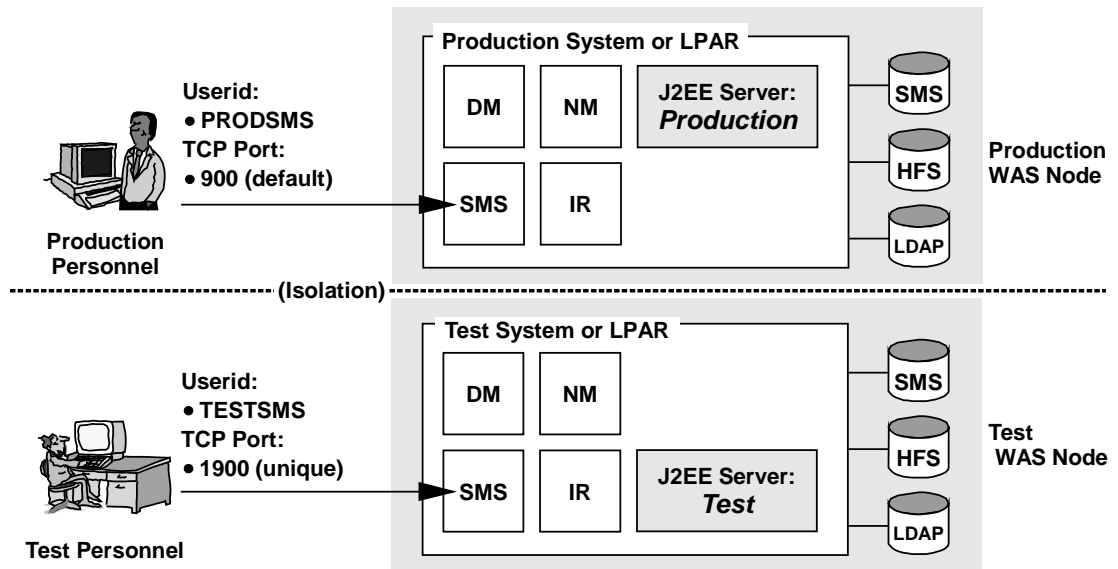
- The same sysplex name
- The RACF database
- The WLM service policy (however, the *application environments* are distinct based on the unique server names you've provided)
- The RRS logstream
- ENF signals and GRS enqueues
- The system console

Detail on "Degree of human isolation"

Because the test WAS node is within the same sysplex as the production WAS node, and because the assumption was the RACF database would be shared across all the systems in the sysplex, the degree of human isolation is less than the full rating. But if your test and production environments for other subsystems are in the same sysplex today, this is nothing new.

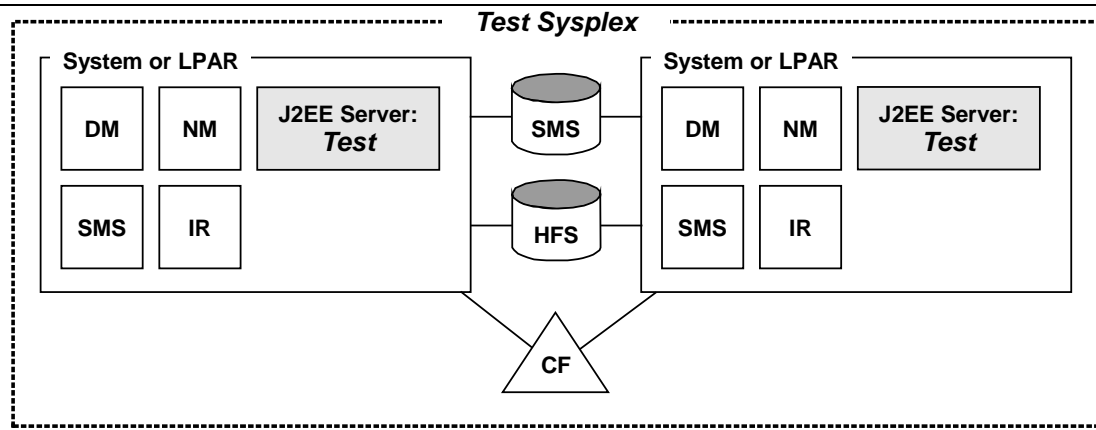
The system console is shared across all systems in the sysplex, so you must be aware of the exposure that may provide to the production environment. Again, if today your test and production is in the same sysplex, this is nothing new.

Because your test and production environments are in different WAS nodes, you can maintain separation of your WAS SMS function, which is perhaps the critical piece of this. As was described under "The Systems Management End User Interface (SMS EUI)" starting on page 17, the way access to the SMS administration function is structured, some conflicts may arise unless you configure in the isolation. This configuration provides that, and the following picture illustrates it:

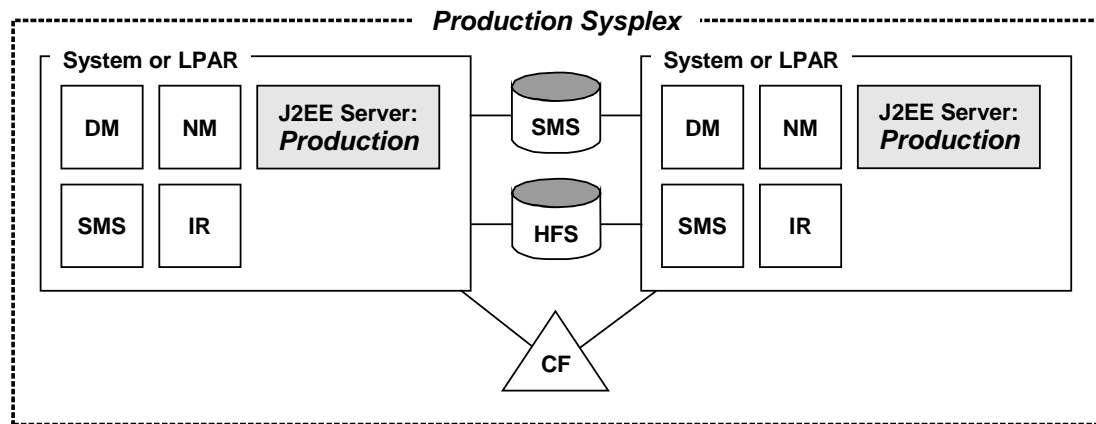


Test and Production SMS function separated by userid and port

Conf #4 - Different WAS nodes in separate sysplexes



Completely Separate — — — — — Completely Separate



Completely separate Sysplex environments; each with its own WAS node

Snapshot rating of configuration

Attribute	Rating (more stars = "better")
Ease of configuration	★ ★ ★
Ability to minimize resource usage	★
Degree of <i>application</i> programming isolation	★ ★ ★ ★ ★
Degree of <i>system</i> programming isolation	★ ★ ★ ★ ★
Degree of human isolation	★ ★ ★ ★ ★

Design Assumptions

This design assumes:

- Two entirely separate sysplexes
- One WAS node configured in each; one WAS node for test, one for production

Whether the test sysplex is a multi-system design or a single system monoplex is really not relevant to this discussion: the key point is the two WAS nodes are completely separated from one another and do not share anything.

Detail on "Ease of Configuration"

This configuration is really no more difficult to configure than is Configuration #1, but the *amount* of work (not the complexity) is multiplied by two. Because the two sysplexes are completely separate, your WAS server names may be identical between the two.

Detail on " Ability to minimize resource usage"

This is rated "one star" simply because it would require the most resource of the four configurations presented here. However, you get the maximum amount of isolation with this configuration. All else equal, this is the *ideal* configuration.

Detail on "Degree of *application* programming isolation"

The highest rating possible. There is absolutely nothing shared between the two environments.

Detail on "Degree of *system* programming isolation"

The highest rating possible. There is absolutely nothing shared between the two environments.

Detail on "Degree of human isolation"

The highest rating possible. The SMS administrative interface is completely separate, and you can be assured your testers have no influence through that tool over the production environment.

Further, the MVS system console is completely separate in this configuration. Configuration #3 did not enjoy this degree of separation.

Appendix A: Naming Checklist

The following provides a template for the creation of your naming convention.

Common Definitions

<i>Byte:</i>	1	2	3	4	5	6	7	8
Control Region Group for ALL Servers								
Server Region Group for ALL Servers								
Unauthorized Userid								
Anauthorized Group								
WebSphere Application Installer Group								
WebSphere Administrator Userid								
WebSphere Administrator Password								
WebSphere Administrator Group								
CTRACE Userid								
CTRACE Group								
CTRACE Proc Name								
CTRACE Data Set Name:	(typically HLQ.xxxxxxxxxx.CTRACE)							

Common Definitions

Base Servers

<i>Byte:</i>	1	2	3	4	5	6	7	8
Daemon Server Name								
Daemon Server Instance								
Daemon Userid								
Daemon Proc Name								
SMS Server Name								
SMS Server Instance								
SMS Control Region Userid								
SMS Control Region Proc Name								
SMS Server Region Userid								
SMS Server Region Proc Name								
IR Server Name								
IR Server Instance								
IR Control Region Userid								
IR Control Region Proc Name								
IR Server Region Userid								
IR Control Region Proc Name								
Naming Server Name								
Naming Server Instance								
Naming Control Region Userid								
Naming Control Region Proc Name								
Naming Server Region Userid								
Naming Server Region Proc Name								
Keyring	(not limited to eight characters)							

Base Servers

LDAP Server

	Byte:							
	1	2	3	4	5	6	7	8
LDAP Userid								
LDAP Group								
LDAP Proc Name								
Administrator user DN, cn=	(not limited to eight characters)							
Authentication ID for DB2 tables								

LDAP Server

Each J2EE Application Server Created

Beyond the initial IVP server, additional J2EE servers are created outside the ISPF customization panels. See "Adding J2EE application servers after initial configuration" on page 12 for more information.

	Byte:							
	1	2	3	4	5	6	7	8
Server Name								
Server Instance Name								
Control Region Userid								
Control Region Proc Name								
Server Region Userid								
Server Region Proc Name								
Default Remote Userid								
Default Local Userid								
Default ID Group								
CBIND CLASS:	(see note)							
CBIND CLASS:	(see note)							
SERVER CLASS:	(see note)							

Each additional J2EE application server

Note: The CBIND and SERVER classes are typically longer than eight bytes, so representing it on the chart would have been awkward. However, the CLASS profile name would typically include the server name as part of the CLASS profile name.

Change History of Document

- March 6, 2002
Original Document
- April 4, 2002:
Updated with explanation on why only one multi-system WebSphere node may exist in a sysplex.
Updated with explanation of multiple environments for servlets, compared that to the WebSphere V3.5 environment, and explained the "simple configuration" (or "alternative configuration") where *just* the WebSphere V4 plugin is used.
- May 16, 2002
Added explanation to picture under Configuration #1 that showed separate DB2 subsystems for Test and Production application data. Clarified point that separate subsystems for Test and Production application data is not a requirement; rather, picture was illustrating effect of doing so when J2EE Data Resources are defined.
Also "tightened up" wording regarding how the "unique JNDI names" issue doesn't really come into play if all one has are servlets.
- August 27, 2002
Added information in Configuration #3 about restriction that WebSphere system tables for Test and Production nodes must be kept in separate DB2 subsystems. The table qualifiers are hard-coded to "BBO" and therefore the tables must have that name. Therefore, only one set of tables in a subsystem can have that value of "BBO."

Index

A

- access control
 - over SMS EUI functions, 10
- application
 - falling back, 15
 - separation within HFS, 18
 - where code stored in HFS, 6
- application JAR files
 - where stored in HFS, 7
- application servers
 - creating additional, 12
- application test
 - contrasted with system testing, 2
- applications
 - copying files in HFS, 7, 9
 - separation of code in HFS, 7

B

- BBO
 - hard-coded qualifier, 29
- BBOWBRAK
 - using to generate RACF script, 13

C

- CBCONFIG
 - and the HFS, 7
- configuration
 - where information is stored, 5
- convention
 - naming, 12

D

- DATASHARING
 - required with Config #3, 29
- DB2
 - as part of configuration repository, 5
 - separate subsystems, 29
- DB2 authorization
 - and server names, 30
- DB2 sharing group
 - and WAS nodes in sysplex, 9
- default JNDI
 - format set by SMS EUI, 14
- deployment process
 - and J2EE resource names in Config #1, 19

E

- EAR
 - placement within HFS, 18

H

HFS

- as part of configuration repository, 5
- how application data is stored, 6
- how server configuration is stored, 6
- location of applications within, 18
- human isolation
 - why still an issue in Config #2, 24
- human isolation
 - explanation of, 2

I

- Isolation
 - what is meant by, 1
- ISPF customization dialog
 - overview of, 5

J

- J2EE servers
 - how server configuration is stored, 6
- java/comp
 - related to JNDI naming, 13
- JNDI
 - different with servlets, 4
 - limitation of hard-coded references, 18
- JNDI names
 - java/comp lookups, 13
 - overview, 13

L

- LDAP
 - as part of configuration repository, 5
- LPAR or system
 - multiple nodes within, 9

M

- multi-system
 - nodes in sysplex, 9

N

- naming
 - checklist for convention, 37
 - overview of naming issues, 12
- nodes
 - configuring multiple in sysplex, 26
 - definition of, 8
 - more than one in a sysplex, 8
 - multiple in sysplex and unique server names, 8
 - multiple in system or LPAR, 9
 - only one multi-system node in sysplex, 9
 - separation of SMS function, 11
 - sharing of SMS repository within, 8

WebSphere V4.0.1 – Test and Production Environments

P

- panels
 - overview of ISPF dialogs, 5*
- Plugin
 - running just V4 plugin, 4*
- programming isolation
 - explanation of, 1*
- properties files
 - to provide JNDI names, 14*

R

- resource names
 - in configuration #1, 19*
- rolling upgrades
 - referenced in Config #2, 23*

S

- separation
 - of application data in HFS, 6*
 - of server configurations in HFS, 6*
- server names
 - must be different in sysplex, 8*
 - must be unique in Config #3, 26*
 - what happens when duplicates discovered, 27*
- Servlets
 - in V3.5 vs. V4, 3*
 - two places where they can run, 3*
- shared JNDI
 - limitations, 19*
- sharing
 - of config HFS, 7*
 - within a sysplex, 9*
- Simple Configuration
 - what it is, 4*
- SMS
 - does HFS update work for you, 7*

SMS EUI

- access control, 10*
- coordinating access, 11*
- default JNDI value set, 14*
- different IDs with Config #3, 28*
- isolation afforded by separate nodes, 11*
- multiple people logging on, 10*
- overview, 10*

SMS repository

- overview of where things stored, 5*

Subsystems

- separate in Configuration 3, 29*

sysplex

- coding multiple nodes within, 26*
- more than one node in, 8*
- only one multi-system node in, 9*
- server names must be different, 8*
- sharing within, 9*

system or LPAR

- multiple nodes within, 9*

system testing

- contrasted with application testing, 2*

T

TCP

- different values for Config #3, 28*

V

V3.5

- servlet execution, 3*

virtual hosts

- need unique in Config #1, 20*

W

WAS

- definition of node, 8*