SHARE Summer 2009
Denver, CO
August 23-28, 2009

Session 1162

WebSphere Application Server Introduction and Concepts for Beginners

Don Bagwell
dbagwell@us.ibm.com
IBM Washington Systems Center

Session Objective and Agenda

Purpose is to explain in plain language what WebSphere Application Server is, what it does, how it works, and why one would consider using it.

Marketing jargon and techno-buzzwords left at the door.

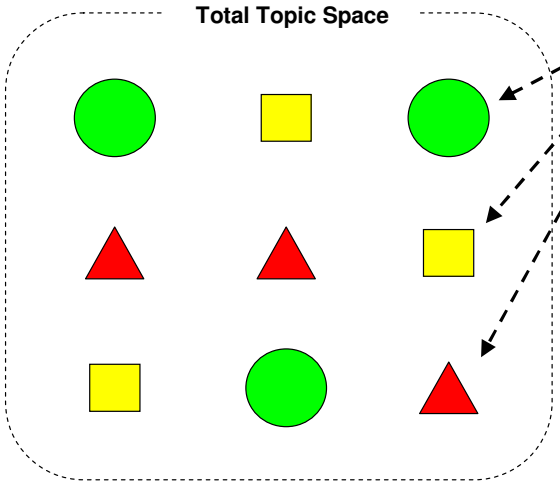
- **Overview of key concepts, terminology and technologies**
  - What an "Application Server" is
  - Java and the role of open standards
  - Review of the kinds of applications WebSphere can and can not run
  - Review of the notion of a "three tier" architecture -- very common with WebSphere
- **High level of WebSphere Application Server z/OS**
  - How it is implemented
  - What it looks like from a system perspective
  - Cells, nodes, servers, clusters
- **Overview of installation and configuration**
  - Understanding the process in simple terms
  - Understanding the keys to success -- in a word: planning
- **Overview of how it's used:**
  - How applications are installed
  - How people access the applications
  - How the applications access data

Introduction Presentations …

This is the agenda of the session. The objective is to introduce you to WebSphere Application Server for z/OS in a way that you understand the key concepts, how it's put together, how you install it and customize it and how to use it.

A Note About Introductory Presentations

In many ways they're more challenging to write and deliver than detailed technical ones:



- **Essential main points**
- **Interesting but peripheral points**
- **Annoying but sometimes important things and exceptions**

Key is to strike the right balance

Be aware that I will sometimes deliberately not mention something because I'm seeking to inform without overwhelming you.

I may also ask that certain questions be taken "offline"

Key concepts …

We have one hour in which to cover a lot of material. And in giving an overview presentation it's important to understand that it is sometimes best to *not* mention things -- that's because some things can create more confusion if given too soon. So the key is to strike the right balance between the key concepts and the somewhat peripheral points so the right picture is painted.

You may already be familiar with elements of WebSphere Application Server for z/OS, and you may notice that some things are not spoken of in this presentation. Please be aware it's because we're attempting to provide broadly acceptable explanation of it all.

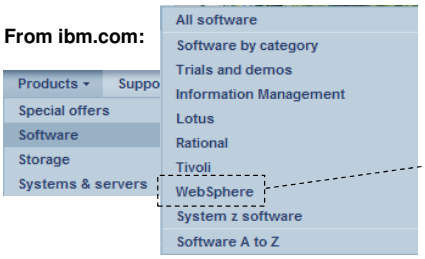
Key Concepts, Terminology and Technologies

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

"WebSphere" is a Brand; "WebSphere Application Server" a Product

We'll start by clearing up a point of confusion about the term "WebSphere."

From ibm.com:



WebSphere Adapters
WebSphere Application Server (Our Focus)
Branch Transformation Toolkit for WebSphere Studio
WebSphere Business Events
WebSphere Business Integration Server
:
:
WebSphere Virtual Enterprise
WebSphere Voice Response for AIX
WebSphere Voice Server

Probably close to 100 products that carry the "WebSphere" brand name

For many, the term "WebSphere" is meant to mean "WebSphere Application Server". Sometimes the acronym "WAS" is also used informally.

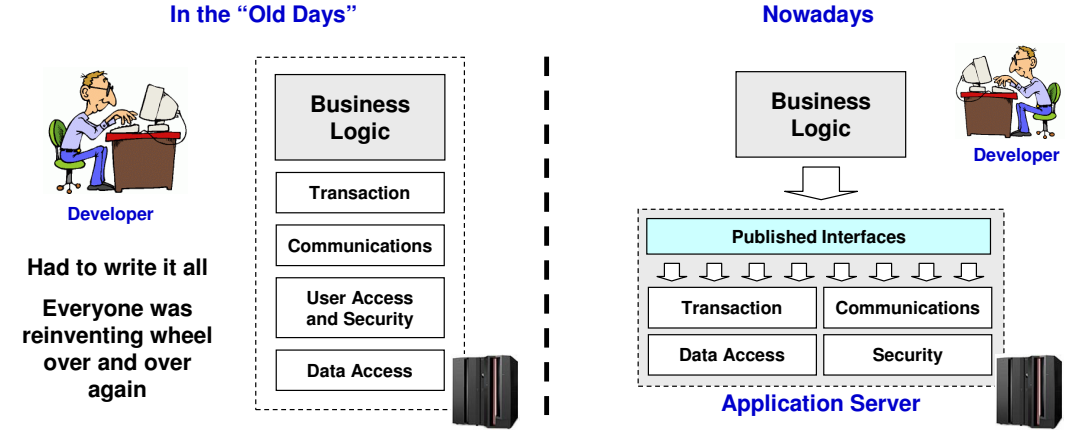
What an "application server" is …

The very first point to understand is that the term "WebSphere" really applies to a family of products, not just one. The more precise way to refer to what we're talking about here is "WebSphere Application Server." If you go out to the IBM website you'll find there are perhaps a hundred products that carry the "WebSphere" brand name.

But, that siad, the term "WebSphere" is commonly used to refer to the application server. That begs the question -- what's an "application server?"

What An "Application Server" Provides

WebSphere Application Server is an "application server" … but what is that?



Purpose is to provide pre-packaged application support stuff so developers can focus on the main business task. No more re-inventing the wheel.

This is not new with WebSphere … IBM had an application server back in 1968!*

So what's the key difference between WebSphere and past application servers?

*** CICS is an application server**

Standards …

So what is an "application server?" It is, as the name implies, a place to run applications. And to understand the role of an "application server" it's worthwhile to go back in time and see how things used to be done.

Way back in the early days applications were written to include everything -- data access code, security (to the extent applications had it at all), communications access, transaction integrity management … it was all written by the application developer. Developers were writing the same kinds of things over and over again.

At some point someone recognized that it would save a lot of time to write those things and provide it as a common infrastructure for use by many applications. That freed up the developers to focus on the business logic. All they needed to do was understand the interfaces to the underlying infrastructure and write their applications to use them.

This is not a recent invention. This basic concept was understood back in the 1960's. CICS is an example of an application server. So too is IMS.

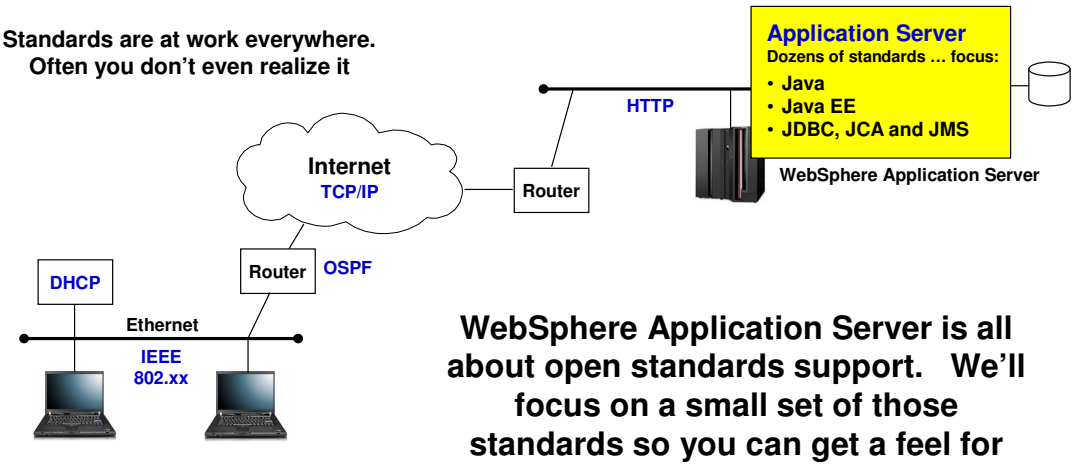
What then is the difference between those earlier application servers and WebSphere Application Server? Answer -- open standards.

Agreed-to Standards

Rules and regulations agreed to by the players in an industry that will allow the different offerings to work better together.

Otherwise we'd have islands of proprietary technology and difficulty interconnecting

Standards are at work everywhere.
Often you don't even realize it



WebSphere Application Server is all about open standards support. We'll focus on a small set of those standards so you can get a feel for the benefit of standards

Java and the JVM …

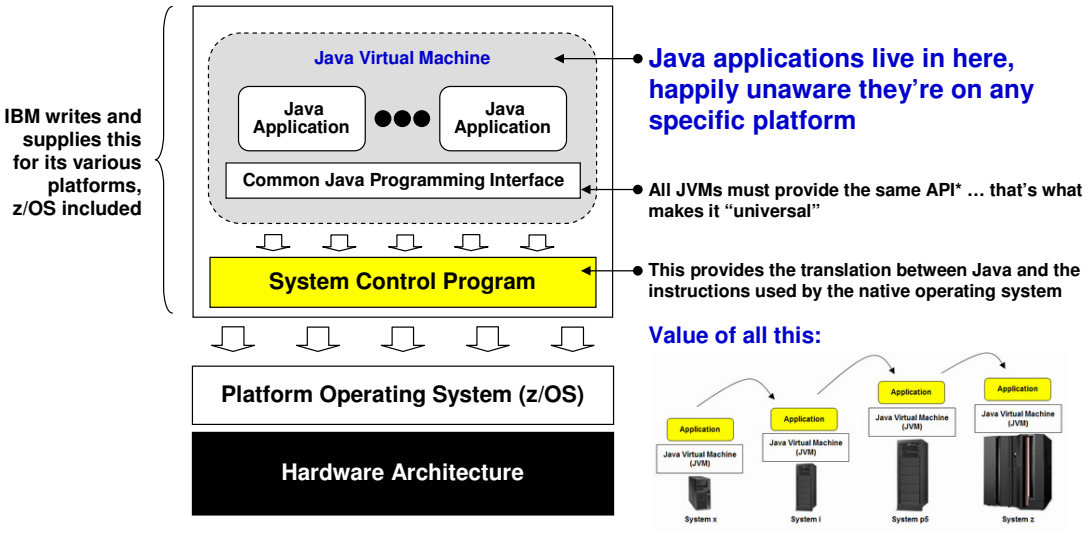
The idea of "standards" -- agreed-to specifications -- is not new to computers, though it's quite common in that realm. In fact, open standards are in use across a broad array things when you do the simplest things with your computer. The chart above shows some of the standards that make using a local area network and the Internet possible.

Within the application server space there are many standards (hundreds, actually), but we're going to focus on only a few -- Java, Java EE and the data connectivity standards of JDBC, JCA and JMS.

Before we go into those standards, the key thing here is to understand that the fundmental objective of WebSphere Application Server is to implement the standards that have been agreed-to by the various open standard bodies. There are many different groups working on standards, and IBM is involved in most if not all of them.

Java and the Java Virtual Machine (JVM)

Java is a programming language; the JVM is what the Java programs runs in. It's what shields the Java program from the specifics of the platform.



The first thing we'll look at is Java, and a very closely related concept -- the Java Virtual Machine.

Java is a programming language originally created by Sun Microsystems. It is an object-oriented language modeled after C/C++. But the focus is not so much on the programming syntax itself, but rather the environment in which it runs. This is the "virtual machine" -- a software implementation of a computer. Java runs "inside" the Java Virtual Machine (JVM), which runs "on" whatever actual hardware is used at the time.

Here's the key -- the JVM comes with something called the "System Control Program," which translates Java programming calls to the native operating system below. That's what "hides" or "shields" the Java program in the JVM from knowing about the underlying operating system and hardware structure.

The Java language is an *interpreted* one -- the Java programs are *not* compiled to the specific hardware instructions. The programs are "compiled" into a kind of half-way state (called "bytecode") that is understood by the JVM, which then turns the instruction over to the system control program, which in turn works against the host operating system.

For the z/OS platform IBM supplies the Java runtime environment, including that layer that translates the Java layer into the z/OS instructions. But inside what IBM supplies for the z/OS platform is the common Java interface … this is what makes Java "universal" across platforms. This is what allows a Java program to run on any platform that supplies the JVM.

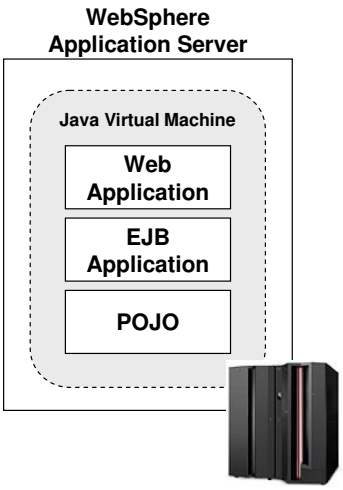
Note: of course, an application may seek data that's available on one platform but not on another. So an application may "run" on any JVM platform, but not "work" if the data isn't there.

WebSphere Application Server for z/OS contains a JVM -- several in fact. And that's what forms the basis for the running of Java programs within WebSphere Application Server.

Different Kinds of Java Programs

WebSphere Application Server can host -- or "run" or "support" -- several different kinds of application -- all written in Java:

It's okay not to understand the details of these things … better at this point just to understand that different kind of programs exist and listen for these terms when others talk about the WebSphere environment.



Web Application

An application that's accessed with a browser. This typically consists of static files (HTML, JPG/GIF), and Java programs that generate dynamic output:

- Servlets -- Java program that contains logic to do things like perform calculations, access data, and format a reply
- JSPs -- stands for Java Server Pages, it's a way to create a dynamic web page that can be populated with dynamic content

EJB Application

Stands for "Enterprise Java Bean," it's a more sophisticated application that's intended for high-end applications. Two flavors:

- Session Beans -- meant to hold the logic of the application
- Entity Beans -- meant to represent data as an "object"

Many EJB applications are made up of just session beans -- easier.

Java EE
Too simple a categorization, but okay for now

POJO

Stands for "Plain Old Java Object." It is the simplest form of a Java program and lately more people are returning to simplicity.
(POJO commonly applies to the EJB 3.0 environment and Java Batch environment)

Data connectivity …

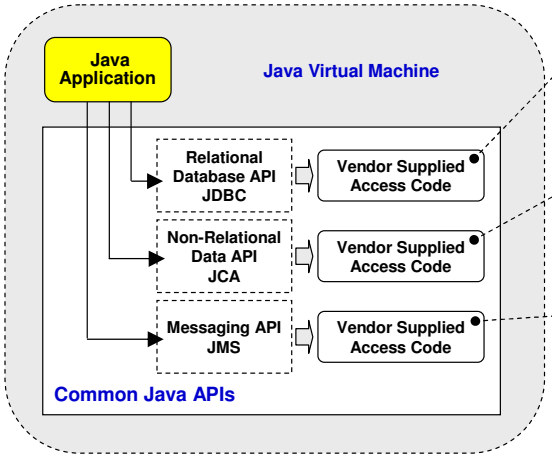
There are several different kinds of Java programs that can run inside of WebSphere Application Server. They all are written in Java, but they take different forms:

- **Web Application-**- this is an application that is accessed by a browser. The most common examples of this are servlets and Java Server Pages (JSPs). A servlet is a Java program that's meant to handle inbound requests and form up a response in return. JSPs are a form of servlet where standard HTML forms a kind of framework for the response, with certain fields designated as "dynamic" and populated at the time.
- **EJB Application** -- this is a more complex structure intended for cases where servlets and JSPs could not do all of what was needed. EJB applications do not interact directly with a browser -- they require a sevlet out front to serve that role. What EJBs provide a more robust design for complex logic. They take two forms -- Session Beans, which are by far the most common, which hold the logic; and Entity Beans, which are intended to be a programmatic representation of data. Don't worry about the difference between Session and Entity beans -- focus on EJBs as a more advanced form of application for enterprise applications.
- **POJO** -- this stands for "Plain Old Java Object," and this is a recent thing in WebSphere. POJOs are simpler structures, and thus easier to write. The newer EJB 3.0 specification has at its core the use of the simpler POJO style.

So we see three main types of Java programs that run inside of WebSphere Application Server. Keep these in mind as we look deeper into how WebSphere works.

Data Connectivity -- JDBC, JCA and JMS

All three are open standard specifications for access different kinds of data. Key is that they provide a defined, standardized interface to "hide" the actual data system where data is held:



Relational Databases
- DB2, Oracle, Informix, Sybase, etc.
- JDBC = Java Data Base Connectivity
- Standard: a defined set of APIs that are converted to the specific commands needed for the backend DB by the vendor supplied code

Non-Relational Databases
- CICS, IMS, SAP, PeopleSoft, etc.
- JCA = Java Connector Architecture
- Standard: a defined set of APIs that are converted to the specific commands needed for the backend data store by the vendor supplied code

Messaging Providers
- MQ as provider, or WebSphere built-in messaging
- JMS = Java Messaging Service
- Standard: a defined set of APIs for applications to send and receive messaging. The actual carrier of the messages can be MQ or the built-in system of WebSphere.

There is more to this when configuring and using it … but this provides the essential point about "hiding" the vendor specifics

Three tier …

Let's now turn our attention to how applications gain access to data. To understand this, it's important to keep in mind of the chief objectives of the open-standard world -- that is, to "hide" the specific complexities behind a standard interface. That exactly what's going on with JDBC, JCA and JMS -- used for relational databaseaccess; non-relational systems (such as CICS); and messaging.

Once again, the key to this is a programming *interface*, behind which is vendor-supplied code to do the actual work. So in the case of JDBC, the interface is provided to access a relational database, but the code to actually do the access is invoked behind the interface. IBM is the supplier of the code that works against DB2; Oracle supplies the code to work against its database, etc.

Similarly, the Java Connector Architecture is intended to provide an interface to non-relational systems such as CICS or IMS, or SAP or PeopleSoft. Once again, WebSphere Application Server provides a way to "load in" the vendor-supplied code to do the intraction, with the application "hidden" behind the standard interface.

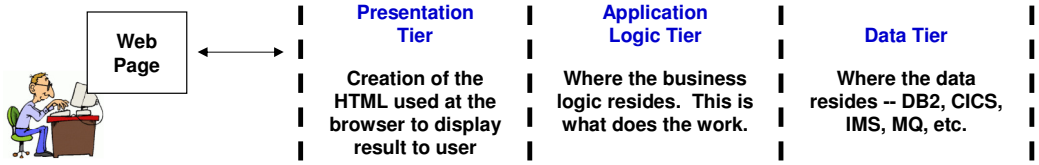
JMS -- Java Messaging Service -- is similar. The actual transport mechanism that passes the message along is provided behind the JMS interface. That's called the "JMS provider". MQ can be used as that provider. WebSphere Appliction Server also comes with a Java-based "default" provider.

To recap -- the key here is that WebSphere Application Server (actually, the Java EE specification, implemented by IBM in the form of WebSphere Application Server) provides a data access interface and the ability to "load in" vendor-supplied code to implement the workings behind the interface.

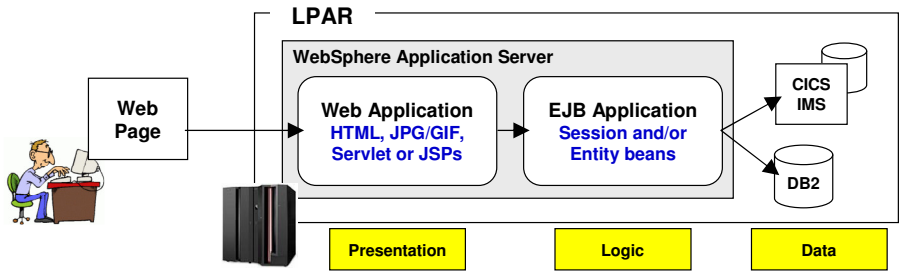
There is considerably more to this, of course … but that is the essence of it.

Typical "Three-Tier" Application Design

We offer this just to put WebSphere Application Server into context with a design concept often talked about and used:



All three *can* exist on the same platform -- logical three, physical one:



Much more to talk about … but this paints the basic picture

What application types run where …

This is a good time to remind ourselves of a key concept employed in the computing industry -- the notion of a "three tier" architecture. The idea is that the three essential "layers" can (and should) be separated to provide greater flexibility. The three layers are -- presentation, logic and data.

Years ago the three were all tightly wrapped into the same ball of code, which made interfacing new technologies very difficult. It was from the lessons learned then that the notion of the three-tier design was born. It was used in applications in CICS, for example, where the 3270 screen formatting was done separate from the business logic.

The same concept applies to WebSphere Application Server -- the presentation layer implemented with web applicatons, with the business logic layer provided by EJBs, and the data later provided by systems such as DB2 or CICS or IMS.

Some suggest the three layers must exist on separate physical platforms, but that's just not the case. It is quite possible to have a *logical* three-tier, but a physical one-tier on z/OS. Many do just this -- the platform provides the ability to avoid single points of failure.

Comparing the Different Places Programs Can Run on z/OS

Let's take a quick look at what kind of programs can run:

	COBOL	Static Web Pages	Java POJO	Servlets JSPs	EJBs
Batch programs launched from JCL	★	⊘ 1	★ 2	⊘	⊘
CICS	★	⊘	★	⊘	★ 3
HTTP Server	⊘	★	⊘	⊘	⊘
"Tomcat" 4	⊘	★	⊘	★	⊘
WebSphere Application Server	⊘ 5	★	★ 6	★	★

Some exceptions and caveats to this … see below

Notes:

1. Is possible to use JCL to launch a HTTP server, but the point here is that batch JCL can't *itself* provide a web server
2. Either BPXBATCH direct invocation of JVM, or the use of something like JZOS
3. CICS supports EJBs, but the specification level supported is quite back-level. In general CICS is not considered the preferred place for EJBs
4. Tomcat is an open-source servlet/JSP engine
5. It is possible to have native code -- C/C++ -- run "in" the WebSphere address space (JNI code). But in general WAS is a Java runtime environment
6. Relatively recent thing in WebSphere

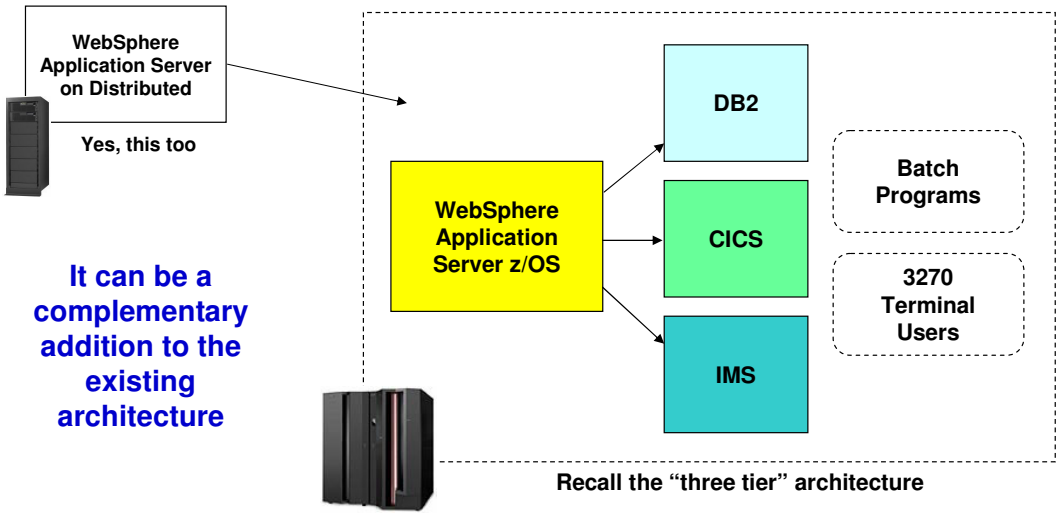
Use in combination …

So a quick survey of the kinds of applications and where they can run -- with the purpose of highlighting the role of WebSphere Application Server. Down the left side you see the different places where applications can run, and across the top you see the types of applications. The bottom row, highlighted in yellow, represents WebSphere Application Server.

The key thing here is that WebSphere Application Server is about running Java-related applications, not traditional applications such as COBOL. Some of the other places to run applications also support Java, but not to the extent WebSphere does. Numbered blocks refer to the notes at the bottom of the chart where various caveats and exceptions are explained.

Use WebSphere in Combination with Other Solutions!

WebSphere Application Server works perfectly well in combination with other traditional systems such as CICS, IMS and DB2:



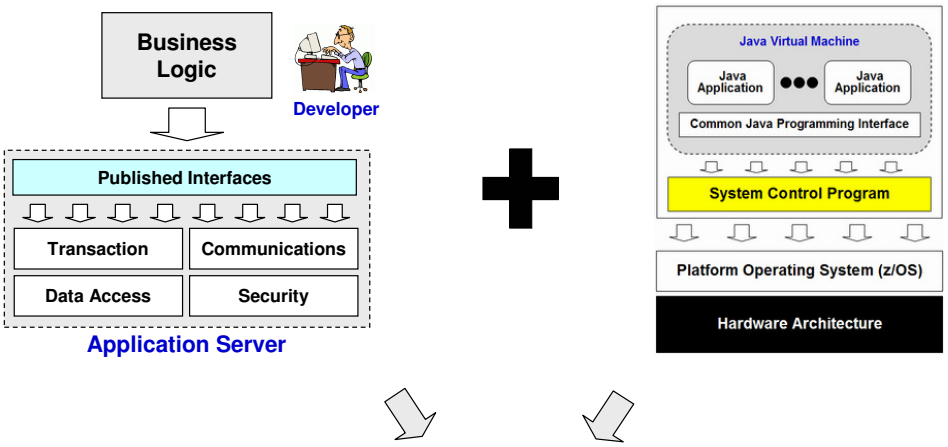
Overview of WAS …

Finally, it should be noted that when considering the role of WebSphere Application Server it is not a case of "WebSphere *or* something else," but more often "WebSphere **and** other things." It should be viewed as complementary to traditional systems such as DB2, CICS and IMS, and in fact may work along with the more traditional ways of accessing those systems, such as batch programs and terminal users.

High Level Overview of WebSphere Application Server

We Need to Marry Two Key Concepts Together

The idea of a framework that provides common services, and the notion of a Java Virtual Machine:



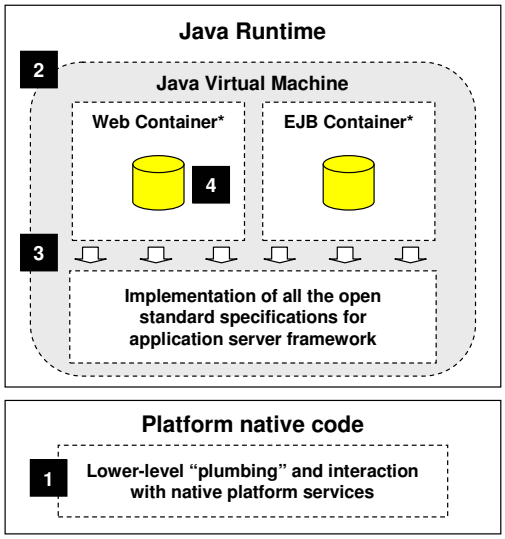
WebSphere Application Server

Now as we get ready to go deeper into WebSphere Application Server, we need to keep in mind that two key concepts given before -- the role of an “application server” and the platform-neutrality of Java -- are to be brought together. That, in essence, is what WebSphere Application Server is.

Schematic Diagram of WebSphere Application Server

Here's a semi-conceptual view of what WebSphere Application Server is:

The real product is of course *far more sophisticated* than this … but this gets the key points across



* "Containers" are just logical software constructs inside the JVM that provide services specific to the type of application that runs in them. "Web Container" is for web applications; "EJB Container" is for EJBs.

1. **Server is Started**
   - On z/OS that's done with a START command (more later)
   - This native code is what establishes the lower-level "plumbing" and allows for the invocation of the Java environment
2. **Java Runtime Established, including JVM**
   - Once the native base is ready, it establishes Java runtime environment and launches the JVM
3. **WebSphere Java Components Loaded into JVM**
   - With the JVM launched, WebSphere Application Server can now load the Java components that make up the Java EE environment
   - This is the "framework" we mentioned earlier
   - This is why WebSphere Application Server is more than just a JVM.
4. **Your Applications Loaded and Started**
   - If they're deployed in the server and configured to start automatically, WebSphere will do that for you.

Now we're ready to see how this is implemented on z/OS

How it's implemented …

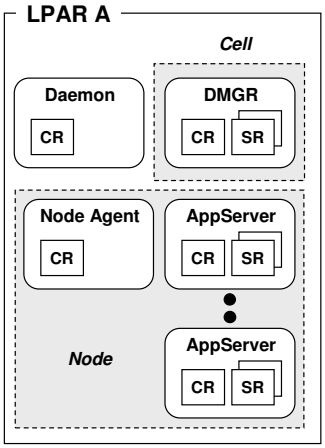
This chart seems busy on the surface, and it does in fact convey some complex things, but the key message is one of how WebSphere Application Server on z/OS provides the Java runtime environment on the platform. Let's walk through the progression so the picture can painted:

1. When a WebSphere server is started on z/OS, the initial code to start is not Java at all, but native code (C++) compiled for the z/OS platform. This is what brings up the intial lower-level framework and plumbing code.
2. Once the foundation is set, then it can fire up the Java environment, including the Java Virtual machine. It is possible to start a JVM outside of WebSphere Application Server, but it would be just that -- a JVM without the other things WebSphere Application Server introduces. Those things are loaded after the JVM is initialized.
3. With the JVM in place, WebSphere Application Server can now load its Java components into the JVM. This is how WebSphere provides the implementation of the various standards and interfaces. This is the "Java EE" framework … which is more than a simple JVM.
4. Finally, with the Java EE framework loaded up, including two "containers" (really just a programming support framework, including interfaces) -- one for web applications and one for EJBs -- your applications can be loaded in. WebSphere will fetch the application binaries out of the configuration file system and load them into the established Java EE environment.

That is the essence of it -- a foundation written in native code that launches the Java environment and then loads in additional Java services and structures to implement the Java EE server specfications. The standards and specifications are common to all vendors -- the way they're *implemented* is up to the vendor.

How It's Implemented on z/OS

The developers of WebSphere on z/OS chose to implement the function of WebSphere Application Server as a series of z/OS started tasks:



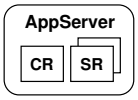
It provides an implementation very familiar to z/OS system administrators

It maps very well to z/OS utilities such as automation and monitoring

But what are those things in the picture?

- **The small boxes inside the curved boxes**
- **CR and SR**
- **DMGR? Node Agent? Node? Cell?**

We'll start with the servers where your applications run:



This icon will be used throughout this presentation to represent the "application server," which is where applications run. The small boxes inside are a design unique to z/OS

Peek inside CR/SR …

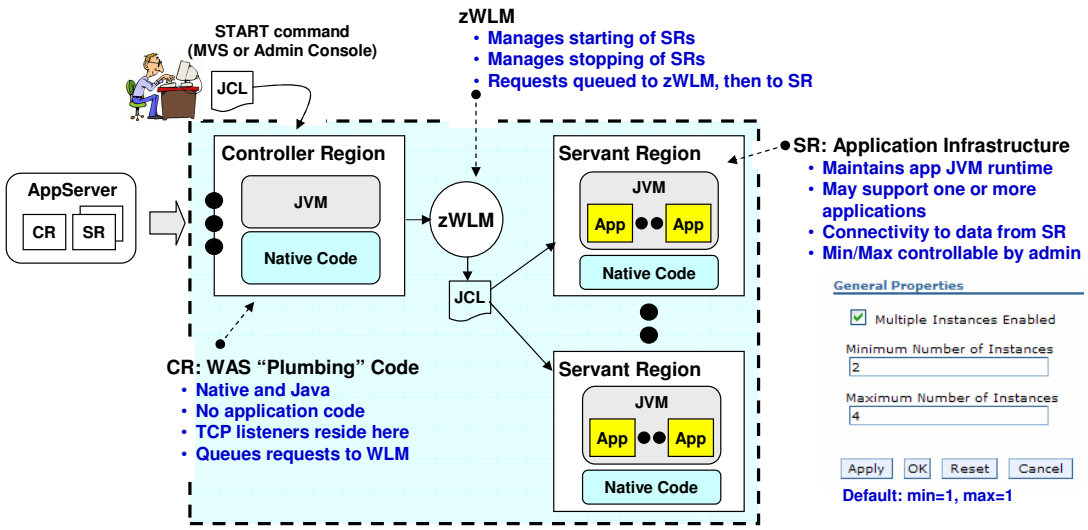
The designers of WebSphere Application Server on z/OS faced a decision -- how to implement it using the available facilities of the platform. They *could* have made it a series of Unix processes managed through an OMVS or Telnet session. But that would have made it a quite foreign thing on the platform … very unfamiliar to most who manage the platform.

So instead they implemented it as a series of started tasks. That makes it familiar at that level with z/OS administrator. And it allows it to map to standard management tools and processes.

The picture above shows a way of illustrating WebSphere Application Server on z/OS -- the curved boxes represent "servers" and the square boxes inside address spaces. But there's much more to explain. And to start, we'll focus on the curved box that represents an "Application Server".

A Peek Inside the Application Server Architecture

We see that inside our little curved-box picture of the Application Server resides two or more address spaces as well as integration with zWLM:



This is a built-in "vertical scaling" mechanism. Also allows for redundancy of application JVM to prevent single point of failure

Exploits platform …

The little icon with the little boxes is not an accident -- it's intended to provide a reminder of the internal architecture of the WebSphere z/OS "application server," which is where your applications run. It is in fact made up of multiple address spaces -- one "Controller Region" (CR) and *at least* one Servant Region (SR). Multiple SRs are possible if you allow WebSphere to start more than one.

Let's start with the CR -- you can think of that as the WebSphere "plumbing" … it's where WAS establishes the communication entry points (TCP ports). The CR is comprised of some native code and a JVM where various Java things are run, but no end-user application runs here. This is strictly IBM-only code running here.

A request comes into the CR first, and it then queues the request to WLM. Yes, z/OS WLM. It acts as an intermediary point between the inbound receipt and the application, which runs in the SR. The request is queued to WLM, and WLM then picks the servant region to assign the request to. If there's only one SR, then the decision is easy. But here's where things get interesting -- if WLM sees that the inbound work is not meeting goals in a single SR, WLM may start a second SR.

Important: but only if you configure it to do this. The default is MIN=1, MAX=1, which means only one SR. But you can change those settings to meet your needs.

What's the value of this? Several things:

- It provides a kind of "vertical scaling" (on the same LPAR) with WLM being the intelligent coordinator of when to fire up additional SR's, but also with WLM being the intelligent coordinator of which SR is best able, at that moment, to handle the request.
- It provides a way to avoid an application outage caused by a JVM failure. When more than one SR is active, physically separate (but identical) JVMs are maintained, with copies of your applications in each. A bad application may cause problems in on SR, but the CR/WLM structure will seamlessly route to remaining SRs.

This is unique to WAS on z/OS. It's easy to get distracted by the specifics of this, but please .. at this point just focus on the multiple address space structure with WLM in the middle.

Exploits the Strengths of the Platform

The WebSphere code base is not completely the same across all platforms. There is a a degree of unique code … to exploit the underlying platform:

Direct Exploitation

Code is written to realize it's on a given platform and unique code is then invoked.

- **zWLM -- managing servant regions to defined goals; internal routing of IIOP based on environment awareness; classification of workload**
- **RMF and SMF -- reporting on transactions and server components**
- **SAF -- security profile repository, including things such as keyrings and digital certificates**
- **Specialty Engines -- zAAPs and zIIPs**
- **Coupling Facility -- for logging**
- **"Type 2" connectors -- native code implementation; true cross-memory communications**

Indirect - Value of "Just Showing Up"

Common code employed, but because it rides on System z and z/OS it benefits

- **Sysplex Distributor and DVIPA support -- for intelligent balancing of traffic (Distributor) and the protection against adapter or TCP stack outage (DVIPA)**
- **Intra-Sysplex or inter-LPAR communications -- XCF (cross coupling facility) or Hypersockets**
- **Sysplex data sharing -- for common data access: DB2, MQ, CICS, IMS**
- **Hardware design -- fault tolerance; redundancy; hot-pluggable, etc.**

The node …

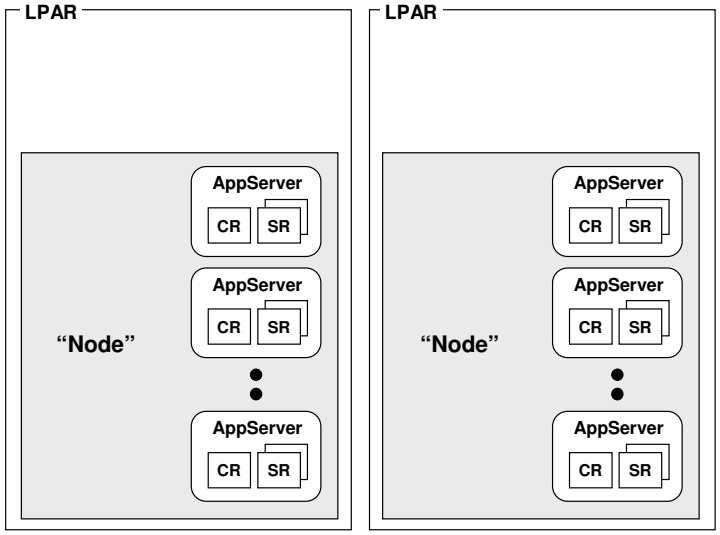
Now is a good time to point out something about WebSphere z/OS … and it relates to the prior chart about how WLM sits in the middle of the CR/SR structure. The point is this -- WebSphere z/OS is not oblivious to the platform on which it runs. In fact, it is quite aware and exploits the platform.

Important: this exploitation occurs in the "plumbing" below the JVM line. Above the JVM line is all about adhering to standards. There, "WebSphere is WebSphere" across the platforms. But below the JVM line is where we get into platform-aware code.

There are two basic forms of exploitation -- direct and indirect. The direct exploitation is like what we just illustrated … where the code makes direct use of z/OS function like WLM. The chart shows other examples of this direct exploitation. The second type is indirect … that is, benefits for "just showing up" on the platform. That exploitation revolves around the inherent strengths of the platform, which provide second-to-none fault tolerance and highly available configurations.

Multiple Application Servers and the Concept of a "Node"

There are many reasons* for creating multiple application servers. A "node" is simply the logical collection of applications servers on an LPAR:



Key Points:

- Nodes are a logical thing … it's not a started task
- They logically organize application servers on an LPAR
- No architectural limit to the number of application servers in a node; limited only by system resources
- Rule: node must stay on an LPAR; it can't span LPARs in a Sysplex

What's the point?

(We'll see in a moment)

DMGR …

* Requirement for separation of application.
Applications have different custom JVM settings.
Different performance requirements

Okay, let's now explore what the other things are in the WebSphere z/OS configuration picture. We'll start with the notion of a "Node." This is actually a cross-platform WebSphere concept. A node is really just a logical collection of servers on a given LPAR. WebSphere maintains the concept of a "node" because that's how it ties servers to a configuration file system, but that's a point we're not quite ready to fully understand. For now, just lock in on the idea of a node being a collection of servers on a given LPAR.

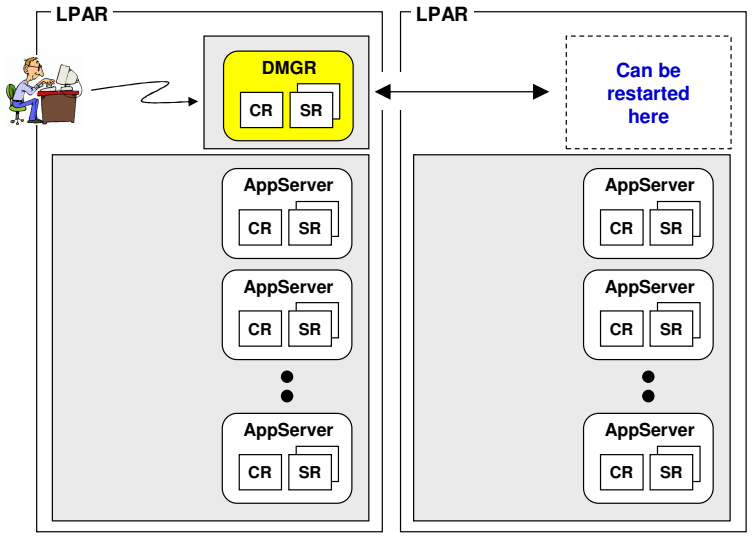
There is no limit to how many servers you can have in a node. It's really a question of resources on the LPAR to support the address spaces.

The rule is this -- a node can't span multiple LPARs. By definition, a node must stay on an LPAR. If you have multiple LPARs in your Sysplex, you would define multiple nodes, such as the picture above illustrates.

Okay … but why? What's the point of this? We'll see that in a moment. But first we have to introduce the "Deployment Manager," which is a special purpose server that runs the Administrative application.

First -- The Administrative Application Server

There is a special purpose server called the "Deployment Manager" that runs the Administrative Console:



Key Points:

- DMGR structure like application server -- one CR and one or more SRs.
- Only the Administrative Console is allowed to run in this special purpose server.
- The Administrative Console is really just a very smart web app that knows how to translate your configuration mouse clicks into updates to XML configuration docs.
- Properly configured, the DMGR can be started on other LPARs
- Only one DMGR is allowed per "Cell" (which we'll describe soon)

Something is missing …

Node Agents …

As mentioned, the Deployment Manager is a special-purpose application server designed to run only one application … the IBM-supplied administrative application. The DMGR server looks just like any other application server with a CR/SR structure.

The Administrative Application, or "Console," is really just a very smart web application that knows how to translate your mouse clicks and keypad entry into modifications to the configuration structure, maintained in XML files. You could hand-modify the XML, but what a mess that would be. First, it would take a lot of knowledge of what XML to update and how, and secondly it would mean a typo could keep things from working right. So rather than force you to do that, the Administrative Console does all that updating-of-XML for you. You see a pretty GUI.

The DMGR is capable of being started on another LPAR if you configure things properly. (How that's done is beyond the scope of this presentation, but it involves the use of Sysplex Distributor.) This provides a way to maintain the configuration capabilities of the Administrative Console during periods of planned (or, let's hope note, but it's a possibility -- unplanned) outages of the LPAR.

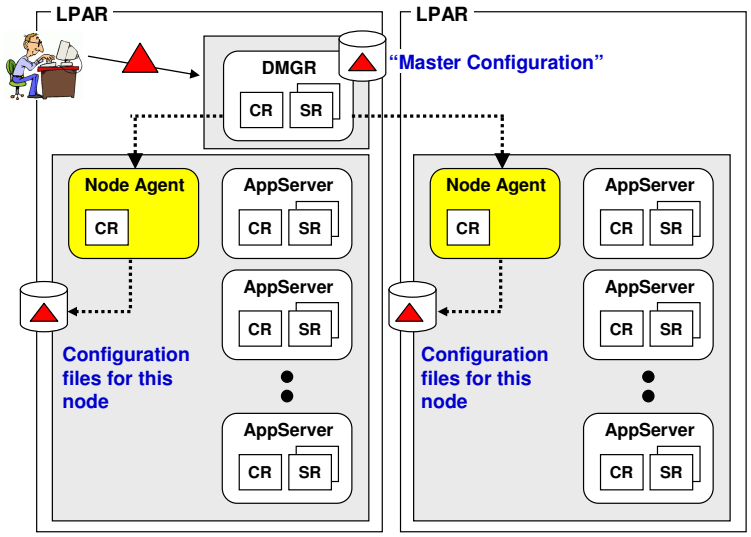
Note: this is a good time to point out that the DMGR is **not** required for the steady-state operation of the other servers in the configuration, including your applications. The DMGR can be down and your applications, running in application servers, can happily continue on.

You can't have more than one DMGR per administrative "domain," or "cell." This is a definitional restriction of WebSphere -- one DMGR per cell. But again, it's restartable on another LPAR and it's not strictly a critical piece of the application serving role of WebSphere. Just configuration updates.

But there's a piece missing between the DMGR and the other servers. How does the DMGR get configuration changes out to the nodes? That's explained next.

Node Agents -- Act on Behalf of DMGR in the Node

Node Agents are single-CR structures that update the node's configuration on behalf of the DMGR, which sends updates to the Node Agent:



Key Points:

- WebSphere is a distributed architecture -- this allows the configuration to be on separate machines and still work.
- This design frees the DMGR from requiring write access to each node's configuration file system.
- Node Agents are just that -- agents that work on behalf of the DMGR to make the changes in the node.
- Act of copying down changes is called "synchronization"
- Trivia - DMGR maintains master copy of configuration, changes made there first, then copied out to the nodes.

Clusters …

The next piece of this puzzle is the "Node Agent." They're shown in the picture as yellow curved boxes. But wait … they only have a CR, but no SR. Is that a mistake? No … Node Agents are pure "plumbing" fixtures -- they only need a CR.

But what do they *do*? As the name implies, they serve as an "agent" for the node. In particular, they are what receives configuration updates made in the DMGR and apply those changes to the node configuration file system. This is done across the TCP network and involves the exchange of updates files from the DMGR down to the Node Agent. This is known as "synchronization."

Could the DMGR do that update without the Node Agent? Well … only if the DMGR had write access to the configuration file system of each node … but the designers of WebSphere did not want to restrict the construction of the configuration where everything had write access to other things. It's a distributed architecture, which means the file systems don't need to be directly accessible.

Note: yes, cross-Sysplex shared HFS or ZFS is possible. But the design is still distributed, and that's why Node Agents exist. Plus, cross-Sysplex write is not a good performer, so the Node Agent structure is still better.

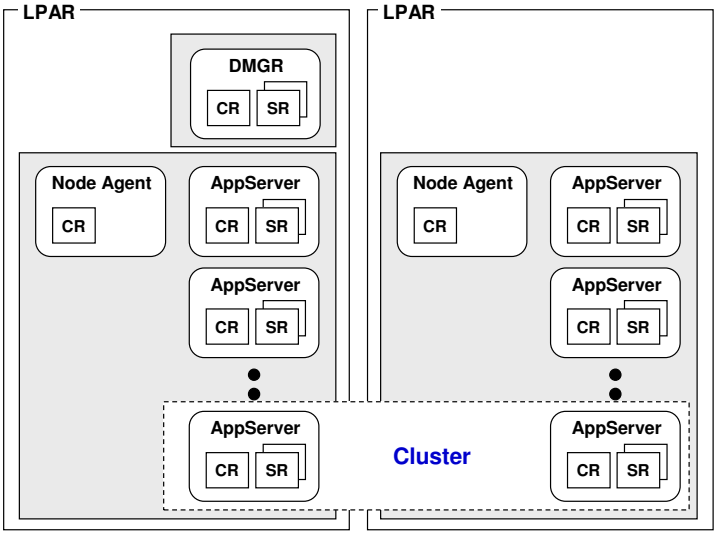
The process goes like this:

- You make changes to the configuration through the Administrative Console, which runs on the DMGR. The DMGR updates its "master configuration" -- which is maintains for the whole "cell" (which we've not yet explained, but think of it as everything managed by the DMGR).
- The DMGR then taps the Node Agent on the shoulder -- "Hey! You have updates." The updates are copied down to the Node Agent in the act of "synchronization." If the Node Agent is not up, the DMGR simply holds the changes and waits for the Node Agent to come up.
- The Node Agent then applies the changes to the configuration file system for the node.

Like the DMGR, the Node Agent is not required for the steady state operation of the servers. It's a configuration update thing.

Clusters -- Grouping of Servers to form a Logical One

WebSphere allows you to define multiple servers that acts as a kind of "single logical server". These are clusters:



Key Points:

- The application servers are in fact separate servers, but WebSphere treats them as one for application deployment
- These are used in HA configurations when mutliple concurrent copies of an application is desired.
- We are intentionally skipping the topic of "front end load balancing" … interesting topic but too much for this session.

Cell …

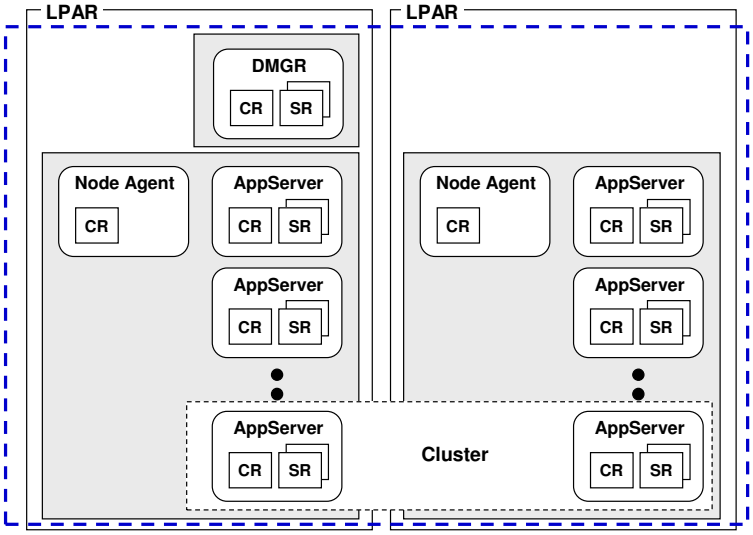
Now we can mention clusters. They are really nothing more than the logical grouping of one or more servers across LPARs for the purposes of providing a highly-available multi-server target. A cluster is a *logical* one, but in fact the servers in the cluster are physically separate servers. The key is that at time of application deployment the cluster will appear as a target, and WebSphere will make sure the application is copied to all the servers (or "members") in the cluster.

Note: cluster could be "vertical" -- that is, on a single LPAR. WebSphere doesn't care. But that doesn't provide the same HA as horizontal across LPARs.

There's a whole topic we don't have time to deal with fully here, and it has to do with "front end" devices to receive and load balance the requests across the servers in a cluster. At the end of this presentation we give pointers to other sessions where this information is provided.

Now We Can Introduce Concept of the "Cell"

The Cell is really nothing more than the extent of administrative control a DMGR has. In this example it controls two nodes on two LPARs … that's the cell.



Key Points:

- The Cell is a logical thing … it is not a started task or address space.
- The Cell marks the boundary for administrative isolation … you can limit who has access to modifications to the Cell. This is how QA, Test and Production is best kept separate.

Answering a few Q's …

Finally we can talk about the "cell," which is another logical thing … it is the extent of knowledge and management control exercised by a DMGR. The DGMR has configuration knowledge of all the nodes and servers assigned to it, and that comprises the *cell*.

Think of the cell as the boundary of administration. It is the best line of separation for the purposes of adminstrative isolation. So, for example, if you wanted to isolate "test" from "production" you may think about separating on the cell level. That would mean each cell would have its own DMGR, which you can then lock down with security access policies so testers couldn't touch the production cell, and vice-versa.

Anticipating Some Questions

May I have more than one Cell?
Yes … no limit to the number of cells you can create.

May I have a cell that spans z/OS and distributed servers?
Yes … but start out with z/OS-only until you gain experience. Then move to the more complicated topic of "heterogeneous cells"

Complication comes chiefly from security issues and the coordination of digital certificates, and the creation of an external userid repository such as LDAP.

What about the Daemon Server?
We intentionally skipped over that to keep things simple ☺

Can an application server belong to two cells at the same time?
No … overlapping of resources like that is not allowed.

You may well have lots of questions at this point. This chart has a few we think may be on your mind.

Installation and Configuration

SMP/E Installation of WebSphere Application Server for z/OS 7.0

Relatively straight-forward SMP/E installation:

```
WAS700.WAS.SBBOEXEC
WAS700.WAS.SBBOHFS
WAS700.WAS.SBBOHFS.DATA
WAS700.WAS.SBBOJCL
WAS700.WAS.SBBOMAC
WAS700.WAS.SBBOMSG
```


`/usr/lpp/zWebSphere/V7R0`

Unix Systems Services
File System

And a short list of relatively simple system programmer steps, all well documented

No module libraries … that's different from in the past. Now the entire product is contained within a file system (HFS or zFS)

This is just the product itself … this is *not* your customized configuration. That's a separate set of sets which we'll cover next.

Customization at hight level …

The SMP/E installation of WebSphere for z/OS is relatively straight-forward thing. A handful of data sets are copied in, with one being an HFS file system which is mounted at a location such as shown on the chart. With V7 the product code is contained within the file system; there are no module libraries as there was in the past.

There's a short list of tasks to do to ready WebSphere z/OS for the environment -- nothing difficult.

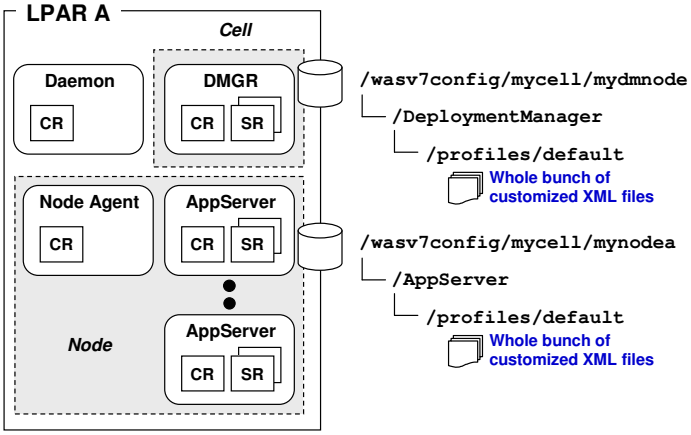
This is really so standard that it's hardly worth going into much greater detail on.

But here's an important point -- this is just the product. This is **not** your customized configuration. That comes next.

Configuration Customization at a Very High Level

The whole objective is to create the configuration information, which is kept in an HFS or ZFS:

Supply names and values so your configuration is customized to your needs → Upload generated batch JCL jobs to z/OS and run → Start Servers



Don't worry what's in those XML files … detail you don't need to know right away

The key is the configuration "tree" needs to get built and populated with your specific information.

That's one piece of the configuration puzzle … what are the other pieces?

Fundamental pieces …

To make use of the WebSphere z/OS installation, you need to create a customization -- a cell structure like we just talked about. At a high-level this is done in a three-step process:

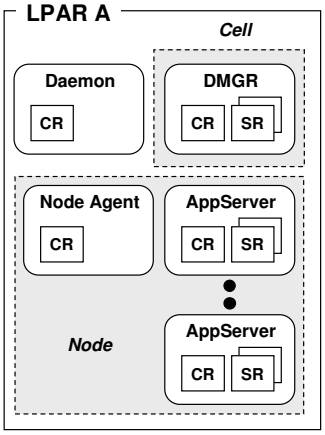
1. Using the customization tool (the zPMT, which we'll speak to in a moment) you supply your customized information -- names, ports, IP addresses, etc.
2. The tool generates some customized batch JCL jobs that are uploaded to the z/OS system.
3. You run the jobs in sequence to create the customization.

The objective of all this is to create the configuration file systems that are what make a node and its servers "real." Without this configuration file system -- full of directories and XML files -- you would not be able to run WebSphere Application Server.

You do not need to worry about what's in those XML files right now. In general you don't need to know what's in there at all … but at times it's helpful to know. But that's advanced stuff and definitely not for this presentation.

Fundamental Pieces of a Customized Configuration

It's helpful to focus on three fundamental things that make up a customized configuration of servers, nodes and a cell:



Configuration File System

- HFS or ZFS, this contains all the XML files that make up the configuration.
- Each node has its own configuration file system

JCL Start Procedures

- This is what is used when starting the servers, Node Agents and Deployment Manager.

SAF Profiles

- They are what provides the essential z/OS security for the started tasks -- Userids and Groups for file ownership and administrative access; STARTED profiles for assignment of IDs, etc.

Two key points:

1. To discard a configuration you don't like, all you need do is clean up these three things (SAF being the most complicated)
2. These things are created by the configuration tool called the "WCT" and are customized with your specific names and values

The WCT …

As we think about what makes up a "customized configuration," there are really three main elements:

1. The configuration file system -- a mounted HFS or ZFS populated with customized directories and XML files. They're customized based on your input.
2. JCL start procedures, which are copied to the PROCLIB you specify.
3. SAF (or RACF by default) profiles.

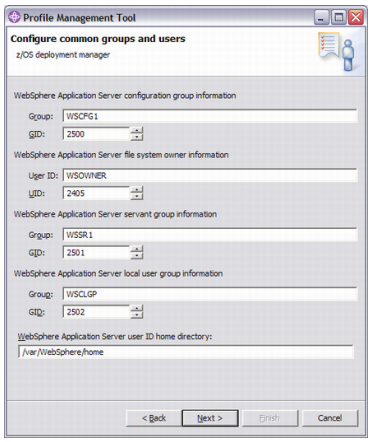
The reason we bring this up is for two reasons:

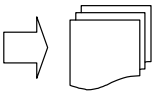
- If you ever need to discard and throw a configuration away -- and we definitely recommend you do that at first as your practicing with this stuff -- it involves reversing those three things: unmounting and deleting the file system; deleting the JCL start procs; and removing the created SAF profiles. The first two are easy; the last is a bit harder but quite possible.
- These things are generated based on your input to the configuration tool, the WCT. The purpose of the WCT is to ask you for a piece of information once, then populate that information into all the pieces in a consistent manner.

Let's look at the WCT tool …

The WCT Configuration Tool

Is a workstation graphical tool that captures key names, values and input from you and consistently imbeds those values in customized batch jobs.





Customized Batch JCL jobs

These get uploaded to z/OS where they're submitted, one after another, to create the configuration runtime.

Uploading and running the jobs is the easy part.

The real challenge is coming up with all the names and values and ports the WCT is going to ask for. Without a plan for those names you'll very quickly get confused.

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS3357 **Spreadsheet …**

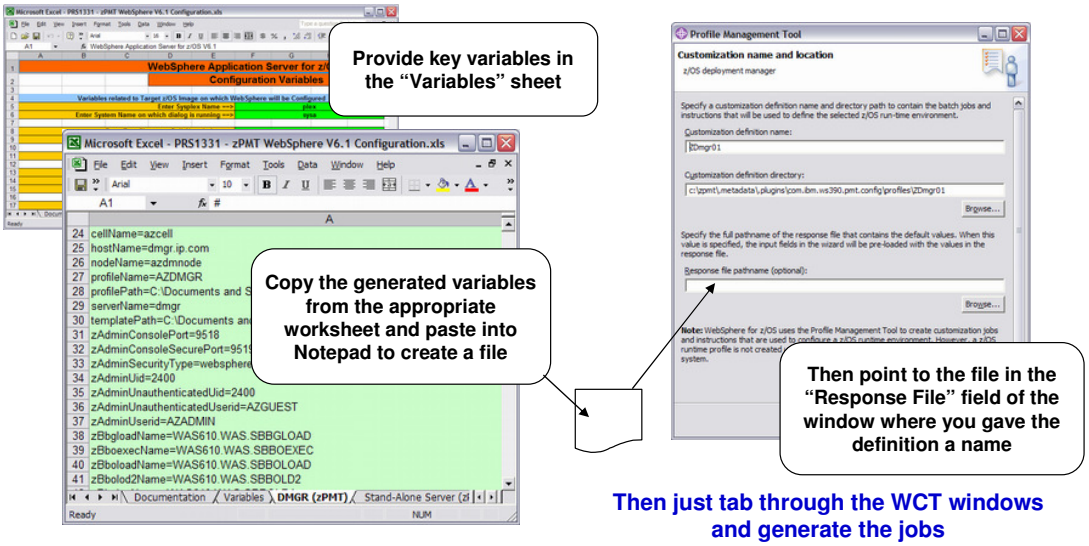
The WCT tool is a workstation-based GUI tool that is designed to capture key input from you and generate the batch jobs that build the customized configuration. The batch JCL is uploaded to z/OS where you run a handful of them to do things like -- allocate and mount the file system, copy JCL to PROCLIB; create SAF profiles, and copy in XML files.

Running the jobs is easy. Standard system programmer stuff. Look for RC=0 and go to the next.

The real challenge is coming up with the names and ports and such to put into the WCT. Without some thinking ahead of time you'll very quickly get lost in the tool. So how do you overcome this? By using the "planning spreadsheet."

The PRS3341 Planning Spreadsheet

An Excel spreadsheet that makes planning values and using the WCT much easier … it helps enforce a disciplined "top down" design:



The purpose of the "Planning Spreadsheet" is to help you lay out the names and values to feed into the zPMT. It can be found on `ibm.com/support/techdocs` under the number PRS1331.

What the spreadsheet does is take from you a small set of key variables, and then from that it constructs all the other names and values. It takes a lot of the work out of planning the input. But as a tradeoff it imposes a pre-set naming convention. It's a convention that has proven itself many times over, but it's a pre-set one based on the programming of the spreadsheet.

The output from the spreadsheet is a "response file" -- a flat set of name/value pairs that you copy/paste into a Notepad session, then load into the zPMPT. And just like that the zPMT is satisfied and it'll generate and upload the jobs to create the customization.

The spreadsheet is probably the single best way to insure a good configuration.

The Generated Jobs and Running Them

Let's look at example of generated job -- this will help "demystify" this:

BBOCCINS	Instruction checklist
BBOSBRAK BBOSBRAM BBODBRAK	Creates the RACF profiles
BBODCPY1	Copies customized JCL procs into PROCLIB.
BBODCHFS BBODHFSA BBOWWPFD	Allocates HFS, creates directory structure, copies in customized XML, and does final build of configuration "profile"

Key Points:

- No real magic to this … just batch jobs that do mundane things
- Key is how the jobs are customized, and that's where the spreadsheet/zPMT comes in
- Running the jobs is easy … making sure jobs have right information is the key

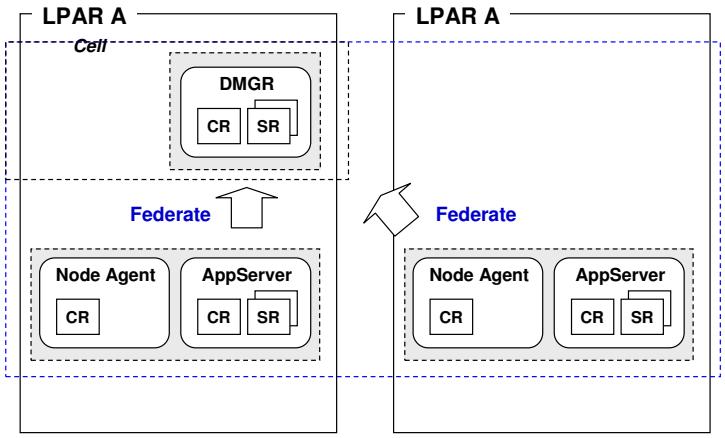
Typical problems are -- typos in the input data (spreadsheet helps avoid this) and insufficient authority (what you need is well documented)

Federation …

The jobs created by the zPMT and uploaded to z/OS look something like this. The purpose of this chart is not to go into detail on each job, but more to demystify this process. The jobs are actually fairly mundane things -- creating RACF profiles, allocating/mounting a file system. The real key is not the running of the jobs, but rather that they were customized based on your input. And with the spreadsheet the planning is much easier.

Build Nodes and Federate

The jobs build a node. To build a bigger cell you do what's called "Federate." This involves running a batch job to join one node into the DMGR's cell:



Key Points:

- **This is a "building block" approach**
- **Build as big a cell as you want using this technique**
- **The DMGR's cell grows to pick up the nodes being federated**

Details of this deliberately left out … don't worry about those right now. Key is the concept of joining a node into the DMGR's cell to make it grow. Get that concept and you're half-way home.

Further customization and usage …

The act of running the customized jobs creates a node. This is a somewhat abstract concept at this point in your learning, but think of the customized jobs as building the configuration file system, which is related to a node.

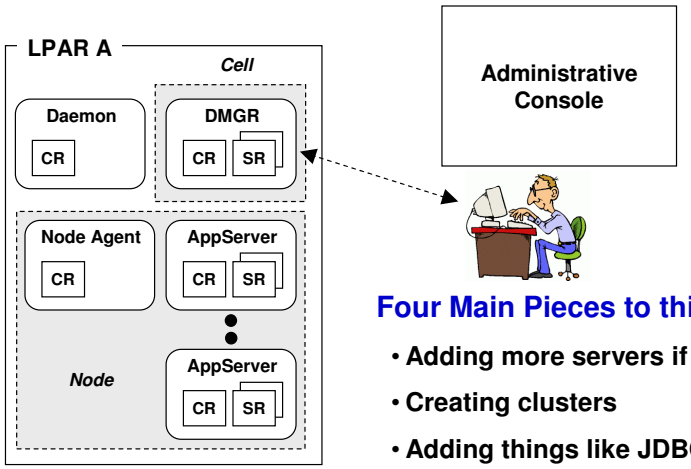
To build a bigger cell it involves customizing a second set of jobs and building a second node, then doing something called "federation." Federation is a fancy term for "joining the node into the DMGR's cell." By doing this, the DMGR's cell expands to include the federated node. You do this over and over to build out the cell to the size you need.

Don't worry about the details of this … it'll come with time. Get the key concept -- the jobs create a node, and the node is joined to the DMGR cell through federation. The DMGR cell grows by this process.

Post-Creation Customization and Using the WebSphere Runtime

What You Have After You've Built Your Cell

After you've done all that you have a configuration that is capable of accepting applications to run:



But your cell will no doubt require more post-creation customization

Four Main Pieces to this:

- **Adding more servers if you see the need for them**
- **Creating clusters**
- **Adding things like JDBC, JCA and MQ**
- **Deploying applications and starting them**

This post-creation customization is common across all platforms … it's not just a z/OS thing.

In fact, it's common across all middleware -- DB2, CICS, MQ … all require some customization

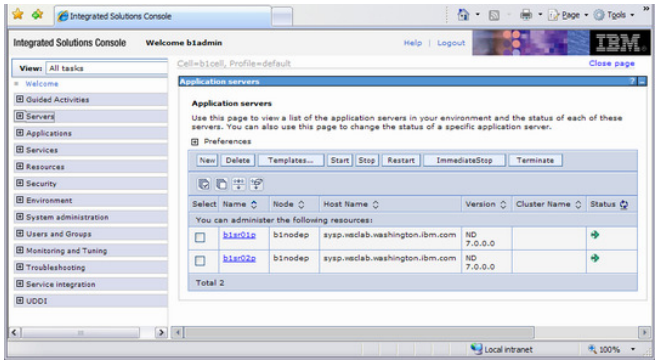
Administrative Console …

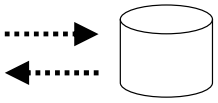
After you've built your cell, you have an operational environment, but it may not have all the things you ultimately need. Here's where the post-creation customization phase comes into play. You may need to do things like what's shown on the chart. And as the chart says, this is not a z/OS-only thing … all platforms of WebSphere require post-creation customization. It's like anything else -- you create it then tweak it.

There are two ways to do this -- the Admin Console or WSADMN …

The Administrative Console

As we said, this is a very smart web application that knows how to update XML files in the configuration based on the point-and-click actions you do





Key Messages:

- **There are a lot of options within the Administrative Console**
- **You learn this over time … you can't master this right away**
- **Always remember what it's doing -- updating configuration XML with your new information, such as JDBC, MQ, applications, etc.**

WSADMIN …

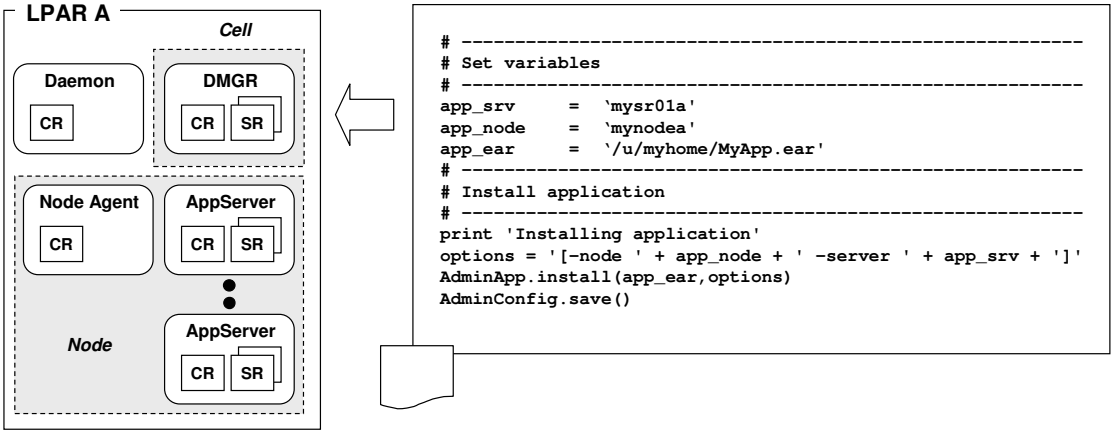
As mentioned earlier, the Administrative Console is an application that runs inside the DMGR and knows how to translate your mouse clicks and keyboard input into updates to the XML files in the configuration. It is an indispensible tool in managing your WebSphere environment.

The Admin Console looks and behaves nearly identically on all the platforms on which WebSphere Application Server runs. The only differences are in the areas of platform specifics … z/OS JCL proc names, for example.

It can be an intimidating thing at first … there are a lot of options. But you'll get it over time. It just takes practice.

WSADMIN Scripting Interface

There's also a programmatic scripting interface that allows you to automate tasks. You can do nearly anything with WSADMIN you can with Admin Console.



Key Messages:

- **Very handy for repetitive tasks**
- **Useful to insure consistent deployments across QA, Test, Production**
- **Does take some getting used to … like any programming language**

Deploying Apps …

An alternative to the Administrative Console is to use the programmatic interface to WebSphere, which is called "WSADMIN." This is a way where you can program "scripts" to do configuration changes. In truth, pretty much anything you can do in the Administrative Console you can do in WSADMIN. For simple on-off things you're probably better off going with the console, but for anything that you'll do over and over again you may want to consider WSADMIN.

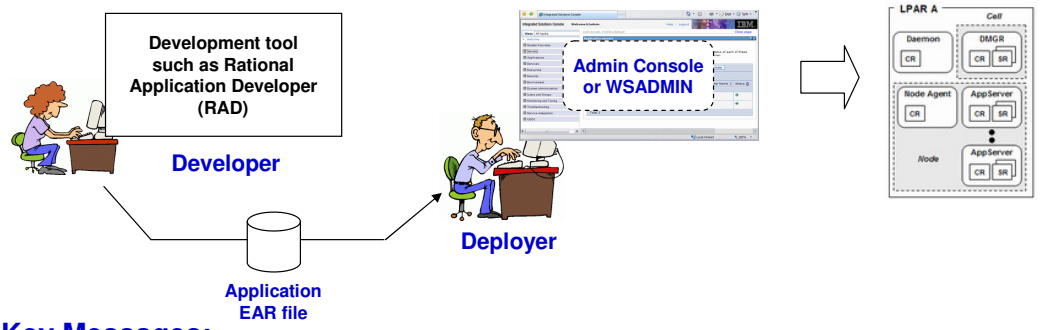
The chart shows a somewhat simple example of installing an application using WSADMIN. It may look complicated at first, but that's because the example above is using Jython (a scripting language) to allow for variable substitution. The heart of the application installation is the `AdminApp.install()` command on the second line from the bottom. That's what does the installation. The rest is really just housekeeping so the variables, set at the top, can be substituted in.

A very common use of WSADMIN is to provide a consistent way to make configuration changes so the exact same action can be done against one cell and then another. For instance, rolling an application from QA to Test is something you'd want to make sure is done the same way in both environments. The Admin Console can be used but you have to be sure you do the same mouse clicks. With WSADMIN you can programmatically script it and insure consistent actions.

WSADMIN is something that requires a little patience to acquire the feel for how it's used. But like any programming language you'll eventually get better at it with use.

Overview -- Deploying Applications

In WebSphere, an application is typically packaged as an "EAR" file -- a zip format file that contains all the piece-parts of the application:



Key Messages:

- **The Admin function will "break open" the zip-format EAR file and put the individual files in the proper places in the configuration file structure**
- **Who performs role of "deployer" is different in each customer … sometimes a separate group; sometimes the z/OS system programmers.**
- **Some knowledge of the application and what's it's designed to do is necessary. You can't deploy applications blindly.**
  Talk to your developers and understand what's going on in the application!

Front-end balancing …

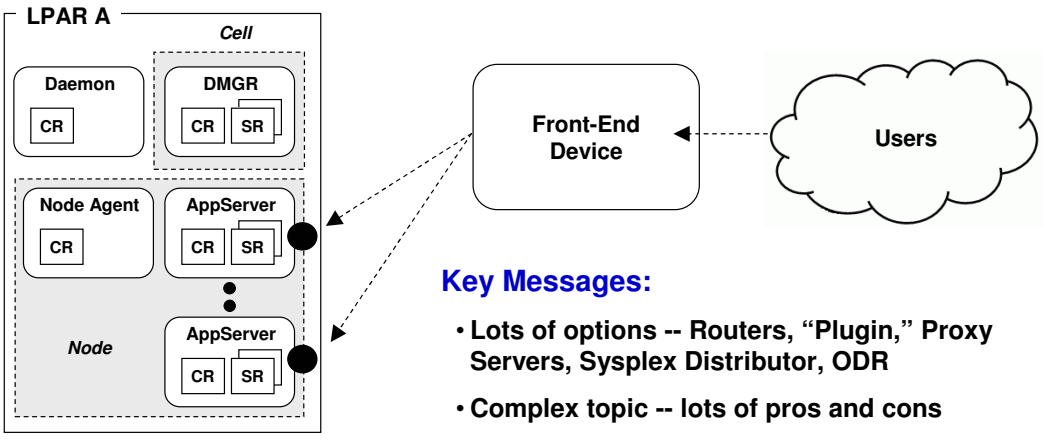
Applications intended for installation in WebSphere Application Server are packaged in the form of a zip-like file called an EAR file (Enterprise ARchive). It is really just a way to contain all the piece-parts for a Java EE application in a single file format for ease of installation.

The EAR is going to be packaged up by the developer (typically), who then turns the EAR over to the deployer. Now the interesting question is … who plays the role of deployer? It varies from customer to customer -- sometimes (but rarely) the developers do that; sometimes it's a system programmer who has taken on the role of deployer in addition to other duties. And in some cases a new role is developed because of the amount of applications being deployed and updated.

In any event the EAR is then deployed into the application server using the Admin Console (or WSADMIN). The EAR is taken in by WebSphere and "broken open" -- the individual files inside the EAR are placed into the configuration file system, and the application's "deployment descriptors" (XML files that provide information about the application's requirements) are read. Much of this is automatic, but some elements of the application deployment will require some knowledge on the deployer's part as to what the application is designed to do.

Front-End Load Balancing Devices

Each WebSphere application server has its own HTTP listeners … so something "out front" is typically required.



Key Messages:

- **Lots of options -- Routers, "Plugin," Proxy Servers, Sysplex Distributor, ODR**
- **Complex topic -- lots of pros and cons**
- **See session 1496, 1:30pm on Monday**

Other topics …

The application servers in a WebSphere Application Server cell each have their own listener ports. You can point the browser straight at those ports, and it does work … but the chances are in a real-world production environment you'll want to design the system to have a "front end" device that intially receives and then distributes the work requests.

There are many options out there.

But rather than go into this topic -- which is somewhat complex -- we'll defer to another session.

Grand Summary

And we come to the end … with a single summary chart:

- **"Application Server"**
  Provides a common set of functions and services so developers can focus on business-value, not plumbing code.
- **"Open Standards"**
  Is what allows the industry to settle on things that allow interoperability.
- **Common at JVM and above; platform specific below**
  This provides the ability to move applications from platform to platform, allowing you to choose the platform based on the strengths of that platform.
- **z/OS Implemented as Started Tasks -- Exploits the Platform**
  Implemented in a way that's familiar to z/OS system programmers. Lower plumbing code takes advantage of the platform while the "JVM layer" shields the applications from having to have direct knowledge of the platform
- **Blended solution -- WebSphere + CICS, DB2, MQ, etc.**
  WebSphere Application Server complements the others, allowing existing applications to be maintained and allowing you to choose the system that best suits your needs.

And the summary chart, which rolls up the major themes we offered throughout this presentation.

End of Document