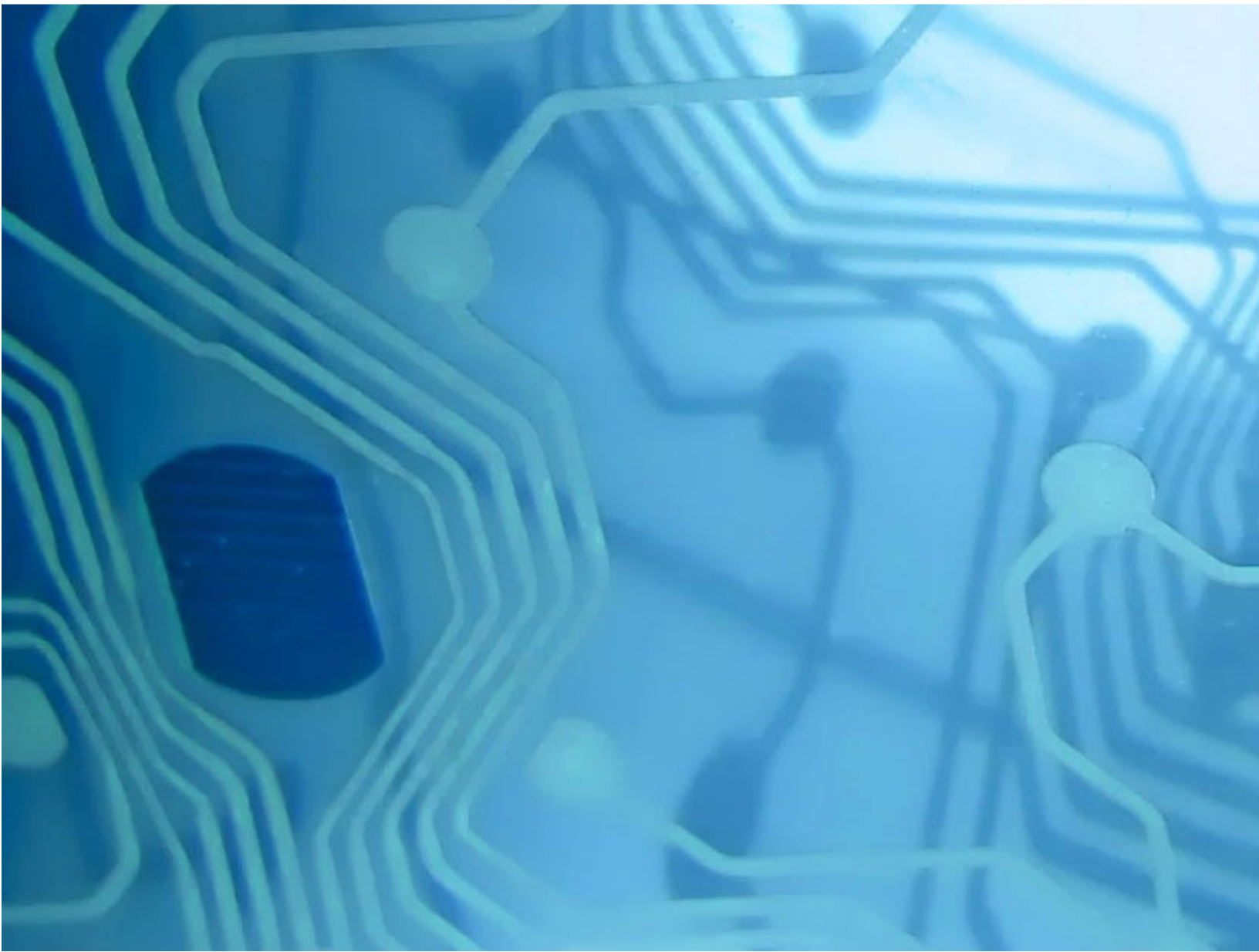
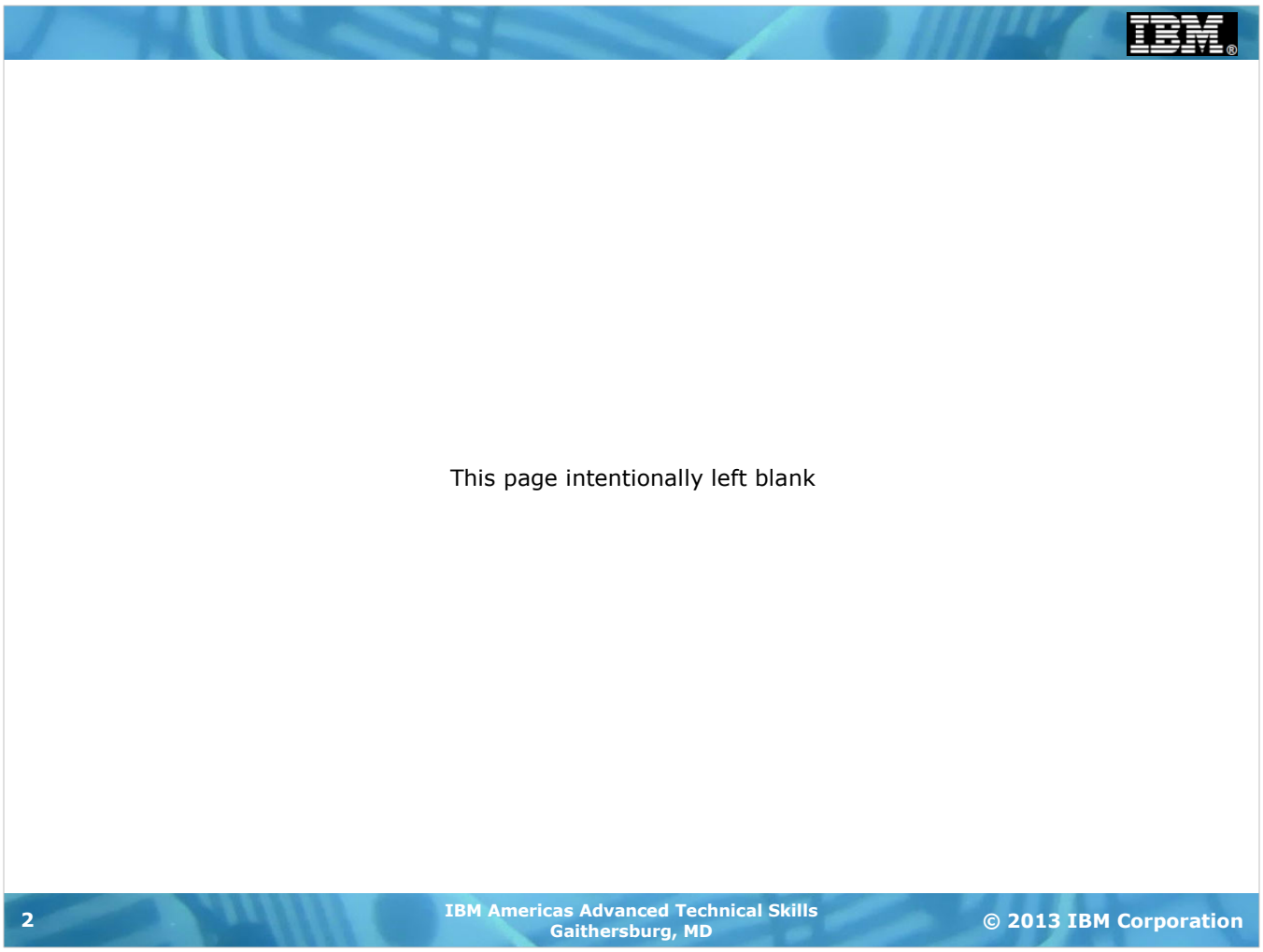
WebSphere Application Server V8.5 for z/OS

WBSR85

WAS z/OS Functions and Capabilities




This page intentionally left blank
2
IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

Agenda

- **Introduction and Overview**
- **Administrative Model**
  - Hands-On **Using the Admin Console, the WSADMIN interface and HPEL**
- **Understanding the Server Models**
  - **Multi-JVM model**
    - Hands-On **Configuring, using dynamic MODIFY, and using WLM to classify work into separate servant regions**
  - **Granular RAS**
    - Hands-On **Extending use of classification XML to control behavior to request level**
  - **Liberty Profile**
    - Hands-On **New lightweight dynamic server runtime model**
- **Access Data**
  - **JDBC and DB2**
    - Hands-On **Type 2/4, the new alternate JNDI failover function, functions unique to WAS z/OS**
  - **CICS**
    - Hands-On **CTG EXCI and the Gateway Daemon**
  - **JMS and MQ**
    - Hands-On **MQ as JMS provider using bindings and client mode**
- **Installation Manager (IM)**
- **WebSphere Optimized Local Adapers (WOLA)**
  - Hands-On **Inbound batch-to-WAS; outbound WAS-to-CICS; HA functions**

Concepts ...

This is the agenda we'll follow for this workshop. We have the workshop divided into five main categories:

- **Overview** -- to set a baseline of understanding and to introduce at a high level some of the new things in WAS z/OS V8.
- **Administrative Model** -- the objective here is to illustrate how in many ways the administrative model is fairly common and consistent across platforms. We'll cover WSADMIN a bit because it's an element of the product that is worth exploring for those who might not be familiar with it. Finally, we'll look at a new WAS V8 function (all platforms) called "High Performance Extensible Logging" (HPEL), which is a new binary log format that has benefits associated with filtering and displaying.
- **Multi-JVM Model** -- the multi-JVM model is perhaps the single most obvious differentiator of WAS z/OS from WAS on other platforms. It's also one of the least understood. It's much more than simply duplicated JVMs. We'll explore in more detail the interaction with WLM, and we'll follow that with a discussion of some new V8 function that offers more granular control of WAS z/OS behavior such as tracing and timeouts.
- **Accessing Data** -- data is the heart of any application design, and a great deal of data is stored on z/OS. In this section we'll review the ways in which key data systems (DB2, CICS, MQ) are accessed, and we'll introduce a new function of V8 that provides failover and failback of data connections.
- **Installation Manager** -- in this section an overview will be given of the new Installation Manager method of creating the hlq.SBBOHFS data set that contains the WAS V8 product file system.
- **WOLA** -- We'll finish with a study of the Optimized Local Adapters (WOLA) function. In a sense this could be considered part of "Accessing Data" but it's sufficiently different that it warrants its own section.

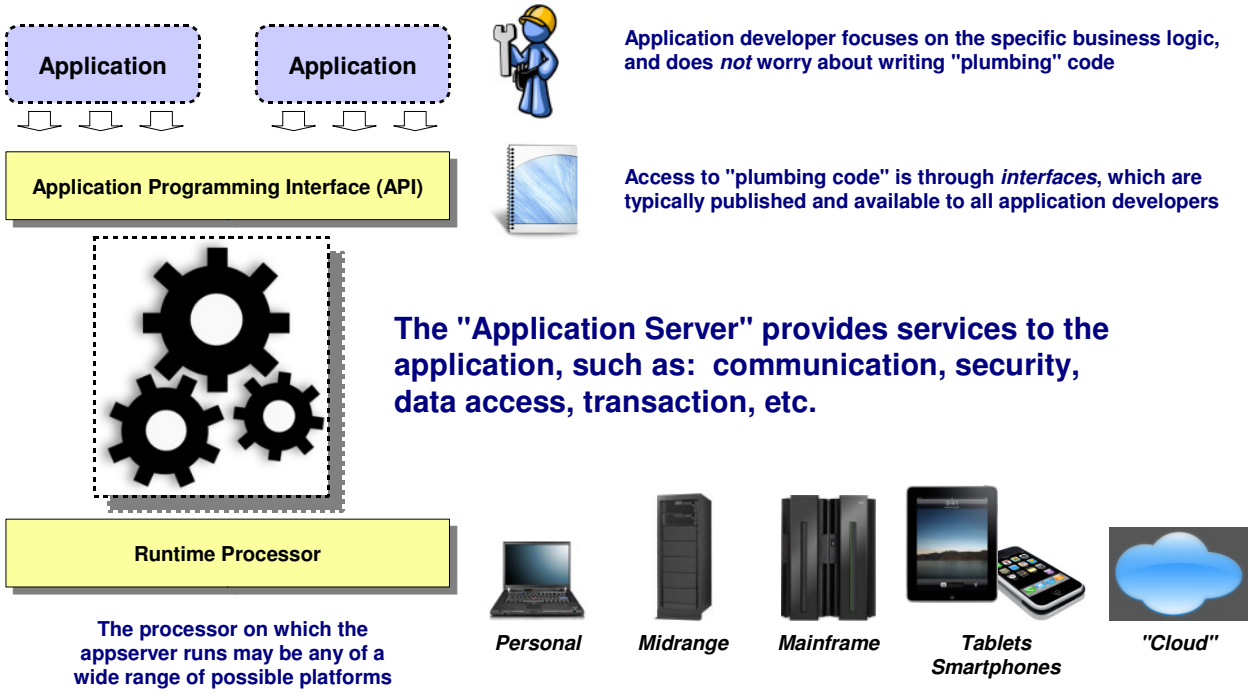
That agenda is supported with a rich set of hands-on labs.

Essential Concepts

We start by getting a few key concepts on the table …

WAS is an "Application Server"

An "application server" provides functions and services to applications so the applications do not themselves do not have to re-invent those functions:



WebSphere Application Server and open standard APIs ...

Within the name "WebSphere Application Server" is the phrase *application server*, and that refers to a computing concept that's been around for a very long time.

Many years ago it became clear that having application developers write all the lower-level "plumbing" code each time was inefficient and created real problems with respect to support and update of that code. A better approach was to write that *plumbing* code -- commonly used functions and services -- once and have applications simply call the functions as they needed them.

To make this easier on the application developer, these functions were provided with programming interfaces that were well-documented. If the application developed needed to check for the authenticity of a user that was logging on, they would check the documentation and see how to call the security function.

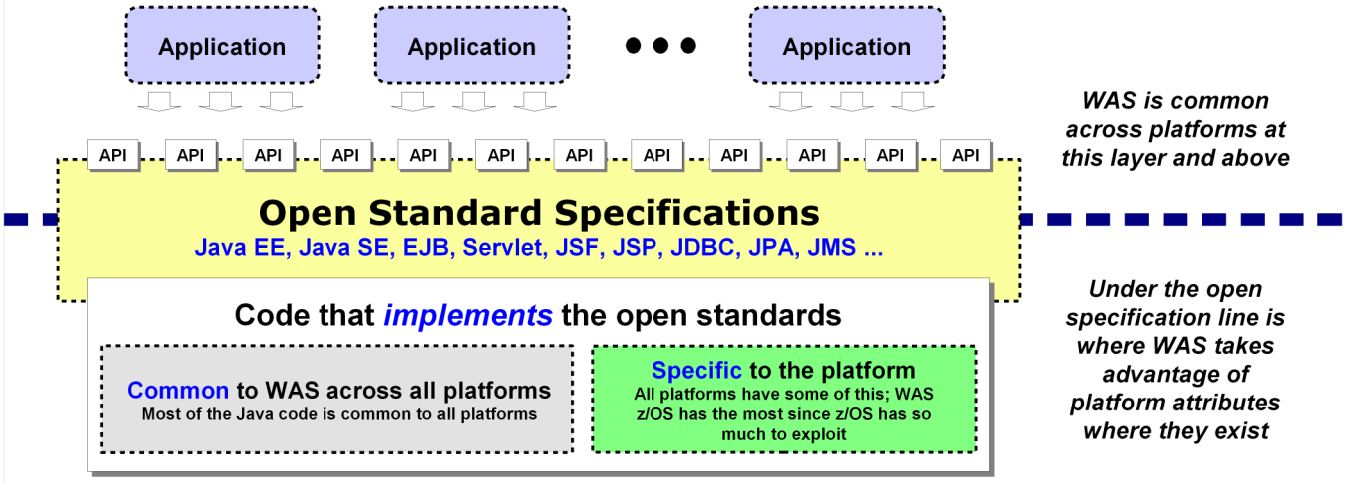
Originally these application servers ran on the most common type of computer available at that time -- the mainframe. An example of an early application server is CICS. Over time the concept moved to other computing platforms as well. Today we see the concept of an application server applying to the full range of what we think of as "computers," as well as relatively new inventions such as the smartphone, tablets, and even "cloud computing."

A development in this space emerged about 15 or so years ago -- a movement to standardize the interfaces so developers did not need to try to learn so many different solutions. From this came a wide range of "open standards" ... standards developed by consortiums of companies and agreed to by all the parties, with the standards then published for all to use.

That forms the basis for WebSphere Application Server.

"WAS is WAS" -- at Open Specification Layer

This is an important starting concept -- it's what makes application development a platform-neutral consideration:



Much of this workshop will focus on what's available to be exploited below the line and how that can be of value

InfoCenter rovr_specs

Java at heart ...

This conveys a very important point about WAS on z/OS compared to WAS on other platforms. Some may think that implies a different programming model, but that's incorrect. The programming model is the same for WAS z/OS as it is for WAS on Windows as it is for WAS on Linux. The exact same Java EE open standards are supported across all the platforms supported by WAS.

This is a very deliberate strategy by IBM. WebSphere Application Server is built on a foundation of Java and open standards. There are many open standards supported by WAS. And the levels for each specification are supported consistency across all the platforms.

Applications developed on one platform can be moved to another and run. That is by design.

The key to understanding how we can say "WAS is WAS" in the same breath as talking about "platform exploitation" is this -- the open standards are really specifications for *interfaces*. Interfaces are implemented with code under (or "behind") the interface that takes the action the interface specification calls for.

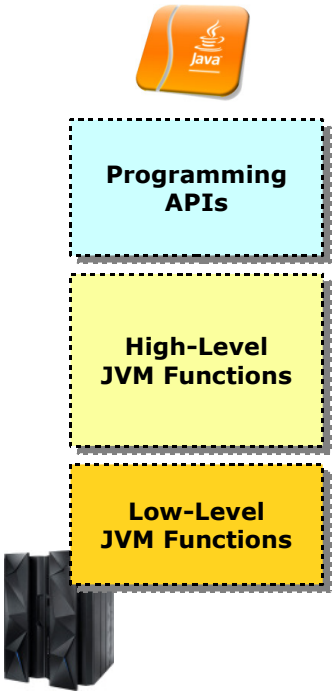
WAS on z/OS has the exact same interfaces exposed to the application as WAS on other platforms. And for a great deal of the code under the interfaces the implementation is also common. But in certain key areas the implementation code under the interfaces takes a detour and exploits key z/OS features. The *benefit* of the z/OS exploitation accrues up to the application. But the applications do not need to explicitely code to any z/OS interfaces. WAS is WAS at the interface level; WAS z/OS exploits below that level.

Much of this workshop will focus on what takes place under the specification layer.

See that little symbol in the lower left? That's our first pointer to an InfoCenter search tag for the article that spells out all the supported specifications in WAS V8 as well as WAS at earlier releases.

IBM Java inside WAS z/OS

It's important to understand that while the Java APIs are industry standards, the *implementation* below the APIs becomes increasingly platform-aware:



SDK conforms to the accepted standards

- IBM SDK provides all the required APIs according to the specification at the level being discussed
- IBM z/OS SDK provides *additional* APIs to take advantage of z/OS platform specific functions (such as SAF security)

JVM Functions common across IBM SDKs

- The JVM is entirely IBM's ... first delivered in 2005
- Many features: generational GC, shared classes
- High-level JVM functions common across IBM Java

System z and z/OS functions

- Takes specific advantage of platform, including exploitation of new CISC instructions available with new System z: z10, z196, EC12
- Big Decimal, Large Page, Out of Order execution, transactional execution, flash paging ... equals *performance*
- Work with z/OS dispatcher to offload to specialty engines

Performance ...

Inside of WAS z/OS is an included copy of IBM Java for z/OS. Three broad points are made on this chart:

1. IBM Java, regardless of platform, is *Java* ... it complies fully with the Java specification.
2. The implementation of the Java Virtual Machine (JVM) under the application specification level is fully IBM's design and development, based on IBM's many years of experience in operating system and virtual OS environments.
3. IBM Java on z/OS is platform aware and takes advantage of specific platform features to provide either additional function or improved performance.

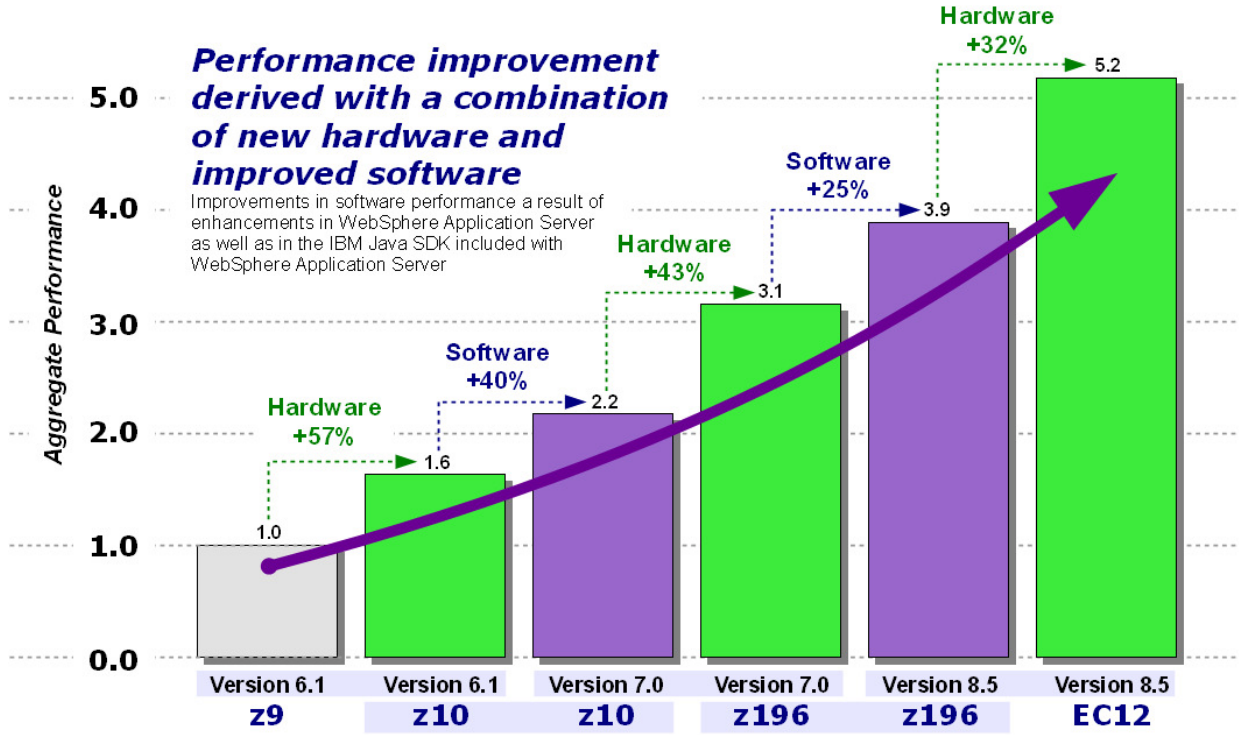
The first point is important because it establishes IBM Java as being *specification compliant* ... applications are written to the particular Java specification level, not to some IBM version of Java. Java is Java at the API spec level.

The second point is important because it establishes the JVM implementation as being something IBM has designed, developed and continues to improve. Provided the application specification interfaces are compliant (which they are), the underlying implementation may be done however a vendor chooses. IBM's JVM is based on IBM's design.

The third point is important because helps to address the question of "Why run Java on z/OS?" The answer is because (a) it performs very well, due in part to the specific exploitation of System z and z/OS features Java on z/OS does, and (b) it offloads to specialty engines.

Performance Over Time

It's a story of improvements in hardware and software:



Controlled test in specific environment. Results vary. This is not a guarantee of performance.

Installation/configuration ...

8 IBM Americas Advanced Technical Skills Gaithersburg, MD

This chart is showing the performance improvements seen in WAS over time with changes in two variables: (1) the version of WAS z/OS and (2) the hardware platform on which the test was run. The trend is obviously upward. The end-to-end improvement is a factor of over five times better performance.

Notes:

- As with any performance chart, this test was performed under controlled conditions with a specific sample workload with a specific system setup configuration. Performance results are a function of many factors, *and your results may vary.* The results shown on this chart are *not* a promise of performance gains and should *not* be taken as such.
- This chart should *not* be used for capacity planning. Other capacity planning tools for System z are better suited for that role. This was a test of WAS z/OS throughput performance under the specified test conditions.
- The test conducted measured achievable throughput when the CPU utilization was pushed to near 100%. The "Aggregate Performance" is a normalized measure of achievable throughput with the left-most bar in the chart serving as the baseline value of 1.0.

Notice how the bars in the chart represent changes in either software or hardware, but not both at the same time. What the chart is showing is how overall performance is a function of both software enhancements as well as hardware enhancements. Combined across the width of this chart and the throughput improvement was 5.2 times better than the baseline test.

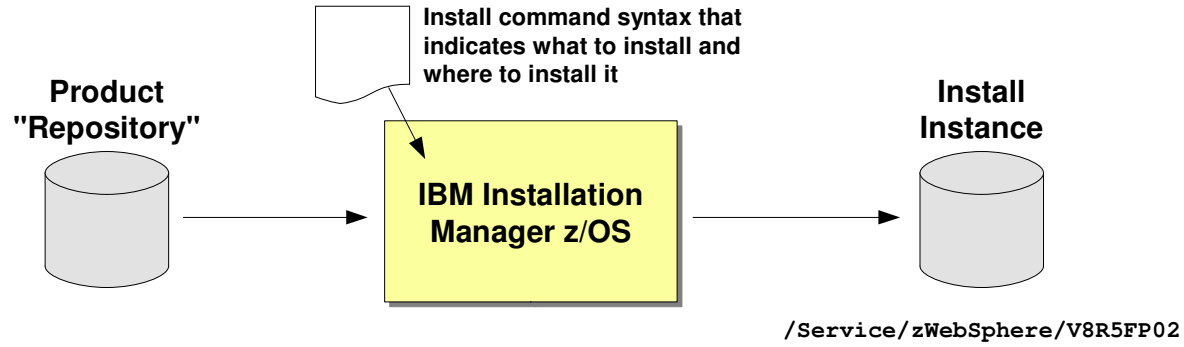
The hardware improvements are well documented on the IBM website for System z hardware -- faster clock speeds, better cache design, and other processor and system design factors. The software improvements came in three forms: (1) improved codepath efficiency within certain parts of the WAS product (particularly the JPA code exercised by the DayTrader sample application; (2) improvements in the efficiency of the JVM, particularly in the area of the Just-in-Time (JIT) compiler; and (3) improvements in the JVM to take specific advantage of new processor instructions introduced in the z10, z196 and EC12 processors.

Installation / Configuration

Setting a high-level baseline of how this is accomplished

Overview of Installation

Unit 5 of this workshop covers the details of this. Here we'll provide a very high-level recap of what's involved to install WAS z/OS:



This can be a local repository or the IBM repository hosted "in the cloud"

On z/OS this can be wrappered in JCL and run as a job that performs the install

Typical practice is to have a separate install image for every version, release and fixpack you want to use

This is a departure from SMP/E. Unit 5 will discuss why IM was chosen for this and what advantages this brings when installing WAS z/OS

Creating runtime ...

Starting with WAS z/OS Version 8 the installation of the product code is handled by IBM Installation Manager, or "IM" for short. This is a departure from the past, where z/OS SMP/E was used to install the product files.

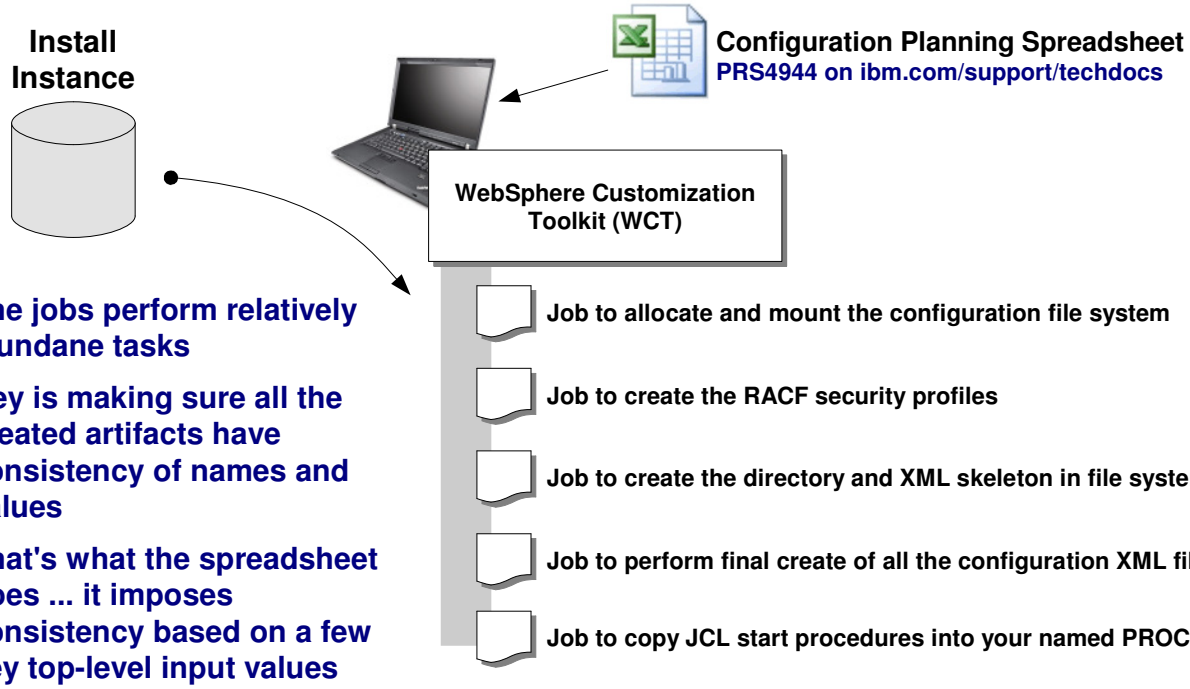
The result is the same -- a file system that contains the product files and binaries -- but the process by which that file system is built is different. In the past SMP/E built the file system; now IM builds the file system. That file system, which we call the "Install Instance" here, is typically created at a `/Service` mount point, and typically the mount point name will carry some indicator of version and fixpack level of the code contained in the file system.

For IM to build this "Install Instance" file system it needs a source from which it will get the product files. IM can't create the WAS z/OS files out of thin air ... it must get those files from somewhere. For IM that "somewhere" is a *repository*, which is a bundle of files produced by IBM. That repository may either be *local* (hosted on a server within your organization), or "in the cloud" (hosted on an IBM server somewhere and accessed over the Internet). More and more the preferred method is to get the files and updates "from the cloud."

IM itself is really just a program that takes commands from you and builds the install instance from the repository you specify. Of course there's more detail to it than that, and in Unit 5 of this workshop we'll go into that detail. The key point here is that *running* IM is really pretty simple once you've done the initial work to set it up and once you've established the input command syntax. Those are the details covered in Unit 5.

Overview of Creating the Runtime

This process has been the same for several versions now. It involves creating a set of customized z/OS jobs, then running those jobs to create the runtime enviornment:



The jobs perform relatively mundane tasks

Key is making sure all the created artifacts have consistency of names and values

That's what the spreadsheet does ... it imposes consistency based on a few key top-level input values

Configuration file systems ...

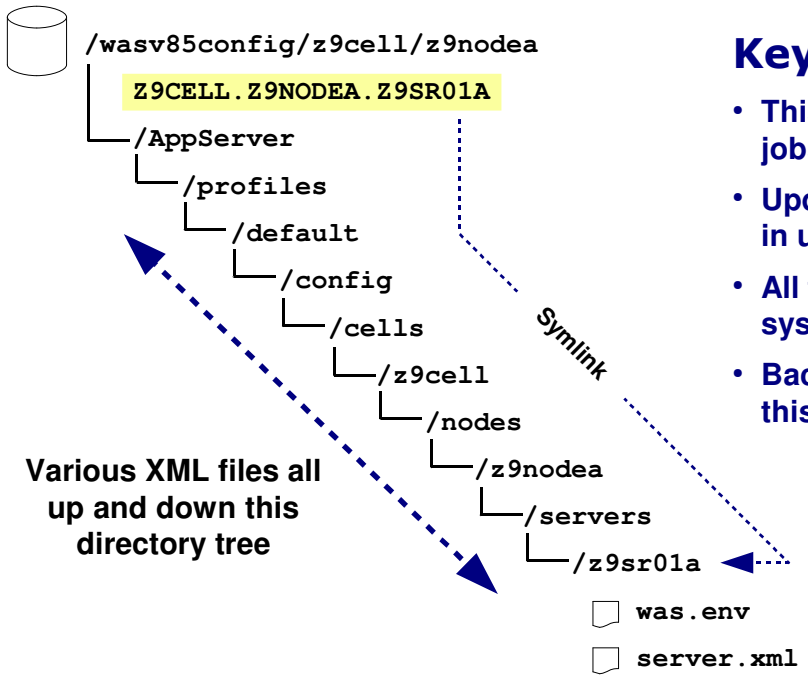
Once IM has been used to install the product files to an "install instance," you may then proceed to building the runtime. This is done in a manner fairly consistent with how it's been done since Version 6.1:

- The WebSphere Customization Toolkit (WCT) is a graphical workstation tool that takes input on names and values and generates a set of customized JCL batch jobs that build the runtime. The WCT has an upload facility that transfers the generated files to the target z/OS system you specify.
- The input for names and values could be entered manually, but a simpler method is to use the planning spreadsheet. The spreadsheet takes a few high-level values from you and then generates all the lower-level variables needed by the WCT. The spreadsheet makes things much easier, and it provides valuable consistency between all the various names. Using the spreadsheet is a recommended best practice.
- The jobs that get uploaded do fairly mundane things like allocate file systems, copy files and generate XML. At the end of running those jobs what you'll have is a mounted file system populated with customized directories and XML, a set of SAF profiles under it all, and JCL start procs used to start the servers.

If you've gone through this process in the past for V6.1, V7 or V8, then the process in V8.5 will look very, very similar. A new copy of the WCT is needed for V8.5 as well as a new edition of the spreadsheet. Otherwise, the process is identical.

Overview of the Configuration File Systems

The configuration file systems contain directories and XML files that represent the runtime. Your customization ends up as changes to these directories and files:



Key Points:

- This is built by customization jobs
- Updates in Admin Console result in updates to various XML files
- All this is contained in a UNIX file system
- Backup and restore is done at this level

Starting and stopping servers ...

At the heart of a runtime environment is a UNIX file system populated with directories and XML that represent all the customization you provided at time of construction and and any updates you did after the runtime was built. Understanding all the details of this file system is *not* important. At first it's enough to know that the file system consists of a bunch of directories and files, and that the administrative console application understands everything and knows where to put things.

As the chart indicates, the key points are:

- The file system is built when you run the customized jobs mentioned on the previous chart. One of those jobs allocates and mounts the file system, and other jobs populate and customize the file system.
- Every time you go into the Admin Console and make changes, those changes turn into updates to the configuration file system. Some updates are simply changes to XML files, and other updates involve creation of new directories and adding XML files. The Admin Console understands all this so you don't have to.
- Since these file systems are contained entirely within the UNIX file system support of z/OS UNIX Systems Services, that means it is contained within a z/OS ZFS file system. That means it can be backed up and restored using standard z/OS utilities.

The other thing shown on this chart is a UNIX symbolic link that's created right up under the mount point for the configuration file system. That symlink consists of the cell, node and server *short* names. Short names are a way WAS z/OS overcomes length limitations in z/OS. Once such length limitation is on the `PARMS=` value in a JCL start procedure. To start a server it's necessary to point to the `was.env` file for that server, and that file is located way down the directory path. The length of that path is quite likely too long for `PARMS=`. Therefore, symlinks are created right under the mount point to provide a short-cut to the `was.env` file for each server. The chart shows only one symlink for one server (server "z9sr01a"), but if you had other servers you'd see a three-part uppercase symlink for each.

Starting and Stopping Servers

WAS z/OS servers operate as started tasks. Standard MVS START commands are used:

```
S Z9ACRA,JOBNAME=Z9SR01A,ENV=Z9CELL.Z9NODEA.Z9SR01A
```

JCL Start Procedure

ENV= is a pointer to the symlink that resolves to the server directory. This provides a way to overcome length limitations in z/OS for the PARMS='' string

One JCL proc may be used to start different servers in the node ... simply by passing a different ENV= string

Key Points:

- **In z/OS environment this is largely "business as usual" processing**
- **The server comes up as a started task (multiple address spaces as you'll see)**
- **It is possible to use supplied `startServer.sh` and `stopServer.sh` shell scripts (those end up issuing MVS START and STOP under the covers)**
- **Also use Admin Console to start and stop certain servers**

Administration overview ...

WAS z/OS application servers are started tasks. They manifest as multiple address spaces (as we'll see in a moment). The key point here is that on z/OS the servers are started as started tasks. That means they can be managed by standard z/OS operational tools such as automation routines.

The syntax of the START command is shown on the chart. The JCL procs needed for WAS z/OS are copied into the proclib you name when you do the spreadsheet / WCT processing. The JCL procs are customized by the WCT and copied into your proclib. Another job creates the SAF STARTED profile to allow the START command to be issued.

The JOBNAME value may be anything you wish, but the recommendation is it be equal to the server short name of the server being started. That allows the server names and job names to line up nicely.

The ENV= string is what points to the specific server you wish to start. The JCL start procs are somewhat generic in that they allow any server in the WAS z/OS "node" ("node" is something we've not yet discussed ... in short, a node is a logical collection of servers configured for a given z/OS LPAR). The ENV= string is what points to the symlink we discussed on the previous chart. As you recall, the symlink is a shortcut down to the was.env file for a specific server. Therefore, by specifying ENV= on the START command, you tell z/OS which symlink to use and with that which was.env to pick up. That starts the specific server you request.

It's also possible to start servers from the WAS Admin Console. In that case what happens is WAS z/OS formats up the MVS START command and issues that on your behalf. Some servers can't be started this way. For example, the Deployment Manager server, which runs the administrative console application, can't be started from the Admin Console because if the Deployment Manager isn't started then there's no Admin Console.

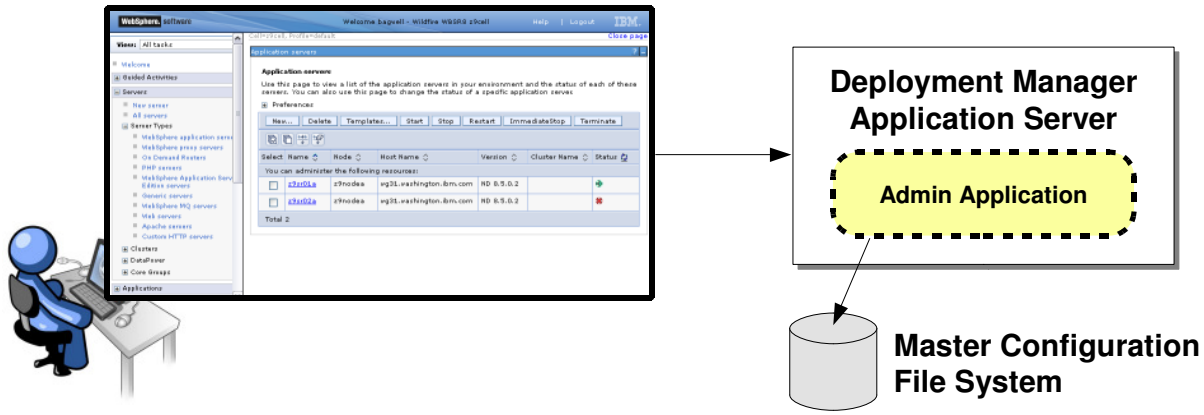
Those familiar with WAS on the distributed platform will know about UNIX shell scripts that start and stop servers. Those work on WAS z/OS as well. They end up doing what the Admin Console does -- that is, formatting up the MVS START command and issuing that on your behalf.

Administration Overview

High level of the Admin Console and administration of runtime

The Deployment Manager and Admin Console

The Deployment Manager is an application server with a dedicated purpose: to run the Administrative Console application:



The Administrative Console's role is to turn your mouse clicks and keystrokes into the appropriate updates in the configuration file system XML tree

Nodes, Node Agents ...

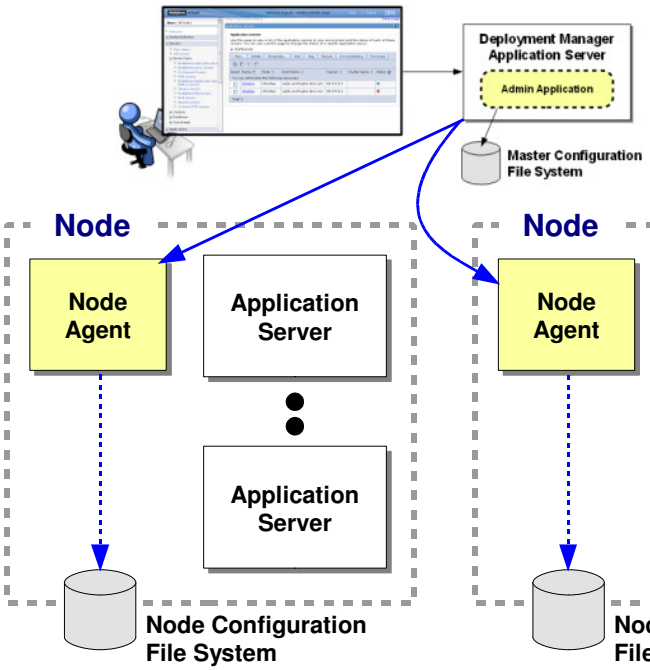
The Administrative Console is a graphical interface to the management application that runs in a special purpose application server called the "Deployment Manager." The Admin Console is what allows you to issue simple mouse click actions in the user interface and have that turned into the appropriate XML updates under the covers.

Note: you could make those those updates to the XML files manually, but that is *strongly discouraged*. For one thing, manually updating XML means your environment may no longer be supported by IBM. In addition, some updates require changes to XML in *multiple places*. Unless you know exactly what changes are needed where, there's a good chance you won't do something properly. The Admin Console, on the other hand, knows what to update and where.

The configuration file system managed by the Deployment Manager is known as the "Master Configuration File System." That file system has information for all the nodes and servers in the cell. We'll spell out what nodes and cells are in a moment ... the key point here is that a Deployment Manager has management control over those things, and the UNIX file system managed by the Admin Console (which runs in the Deployment Manager) is aware of *everything* in its environment. Changes are first made to this "master" file system, then copied out to the file systems for each node.

Nodes, Node Agents and Synchronization

Nodes are way of collecting up application servers on an LPAR. Node Agents are a way to get configuration changes from the master configuration out to the nodes:



Nodes are a collection of servers on an LPAR

Admin Console updates the Master Configuration File System

Node Agents copy changed XML files from Master down to the node file system

The cell …

On the previous chart we indicated how the Admin Console makes updates to the "master configuration," and we hinted at those changes being copied out to other file systems. It's now time to explain all that.

WebSphere Application Server (WAS) is designed around the principle of a *distributed architecture*. By that we mean the design assumes the WAS configuration will span multiple operating system instances. That's true on non-z/OS platforms, and it's true on z/OS as well. On z/OS the other operating system instances are typically other Logical Partitions (LPARs) on the same mainframe server hardware. Still, each LPAR is a physically separate instance of the z/OS operating system.

Nodes are simply a WAS concept of logically collecting up application servers that exist on a given LPAR. The picture above shows a hypothetical two-LPAR configuration with a set of servers on one (which is one "node") and a set of servers on the other (which is another "node"). The Deployment Manager is a separate node unto itself and it may run on the same LPAR as an application server node, or a completely separate LPAR.

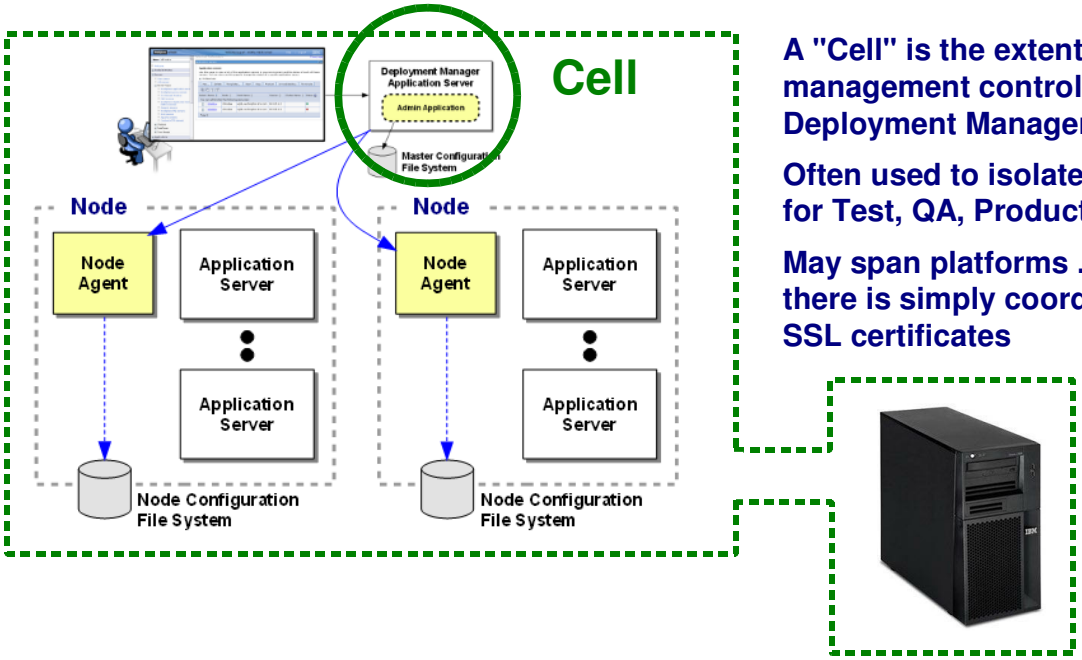
Changes made through the Admin Console are made to the Master Configuration file system, which is different from the configuration file system for each node. So how do the changes made to the master propagate out to the node configuration file systems? By a process known as *synchronization*, which is a fancy word for "updated files are copied out and written to each node's configuration file system."

Rather than assume the Admin Console has write access to file systems on other operating system instances, the WAS architecture includes an *agent process* in each node to act as the intermediary between the Admin Console and the configuration file systems for each node. This process is called the "Node Agent" (creative, huh? ☺) and when an updates are made to the master configuration those changes are copied out to the nodes via the Node Agent.

Note: the act of synchronization can be controlled. By default it occurs at time of change or every 10 minutes, but can be made to occur less frequently, or made to occur only when you manually invoke it. So in a production environment changes can be staged and carefully propagated by your deliberate action to start synchronization.

The "Cell" -- Boundary of Management Control

The "cell" is the extent of management control for a given Deployment Manager. Most run with multiple cells. And a cell can span platforms if you wish:



A "Cell" is the extent of management control for a given Deployment Manager

Often used to isolate security for Test, QA, Production

May span platforms ... issue there is simply coordination of SSL certificates

WSADMIN ...

Now we're ready to define what a "cell" is, even though we've mentioned that term several times before. The "cell" is the extent of management control for any given instance of the Admin Console administrative application, which runs in a Deployment Manager. For any given Deployment Manager the master configuration for that Deployment Manager understands what nodes and servers it managed by the XML it sees in the configuration.

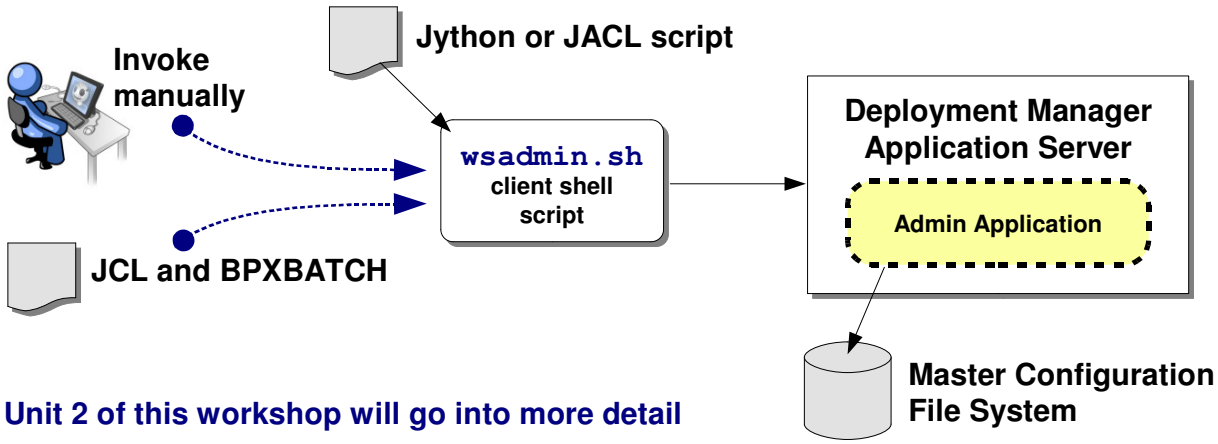
You may have multiple cell environments, and most likely will have multiple cells. Cells are used to separate different environments such as test, QA and production. You may specify different security controls at the cell level, which allows certain people to access the test cell but not the production cell.

The final point on this chart is that a cell may span multiple platforms. It is not limited to "just z/OS". When a cell spans z/OS and other platforms the consideration becomes managing the coordination of security certificates so every platform in the cell can talk to each other. When the cell is entirely on z/OS in a Sysplex, the certificates are stored in SAF and available to every node in the cell. That makes certificate coordination very easy. When the cell jumps off z/OS then the certificates need to be exported and imported so SSL can take place between the various platforms.

Note: a WAS "cluster" can not span z/OS and non-z/OS. The reason is because IIOP routing within a cluster is managed by z/OS WLM, and if the cluster spans z/OS and non-z/OS then WLM can't "see" the whole cluster. So a cell that spans z/OS and non-z/OS is possible, it's just that any cluster you create can't span z/OS and non-z/OS. You may create clusters on z/OS and other clusters on non-z/OS ... that's okay.

WSADMIN -- A Programmatic Interface

WSADMIN is a scripting interface to WAS (all platforms). It provides a way to programmatically perform administration actions:



Unit 2 of this workshop will go into more detail

Anything you can do in Admin Console, you can do using WSADMIN scripting

Allows you to automate common tasks such as application deployment ... which provides *consistency* of actions across Test, QA, Prod

Liberty ...

Up to this point we've referred to the Admin Console web application as the means by which configuration changes are made. But there's another method available to you -- the WSADMIN scripting interface. This allows you to use scripts to programmatically perform configuration actions. In Unit 2 of this workshop we'll discuss WSADMIN in a bit more detail.

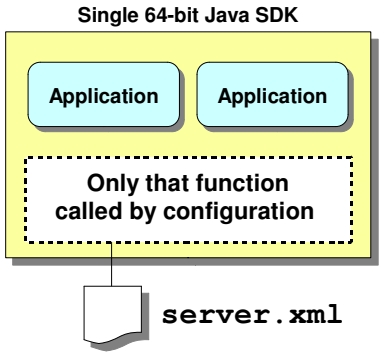
The main point here is that a programmatic management interface provides you the ability to enforce *consistency* of configuration actions ... either for repeated operations on the same cell, or for actions across cells such as you might do between test, QA and production.

There are several different ways you may invoke the WSADMIN function. One way is manually from a UNIX shell environment. Another is to wrapper the UNIX command in JCL using BPXBATCH. By using JCL you then open up the possibility of including WSADMIN actions in standard z/OS operational control tools.

Again, more on WSADMIN in Unit 2.

Liberty Profile

The Liberty Profile is a lightweight, dynamic, composable, single-JVM server model. It is offered along with "Traditional WAS z/OS" ...



- **Configuration file determines what functions are loaded**
- **Starts very quickly, consumes much less memory than traditional WAS z/OS**
- **Servlets, JSPs, web applications**
  Updated in V8.5.5 with additional features
- **Dynamic -- change server configuration or applications without server restart**
- **Not part of traditional WAS "cell" or "node" structure**

We have a section and lab on this topic

Applications ...

One of the significant new functions that comes with V8.5 is the "Liberty Profile" ... which is a new server model that provides a single-JVM server. This server is composable, which means you configure only those features your application needs. This helps reduce the overall resource usage. The design is also very dynamic, with changes to the configuration and applications detected and updated automatically.

Right now the Liberty Profile design is limited to web applications -- servlets and JSPs. Those applications do not need the full function of traditional WAS, so deploying Liberty Profile servers is a way to reduce resource usage.

Note: with Liberty 8.5.5 additional features have been added, including EJB-lite, JMX, caching and a list of other things. In the unit on "Server Models" we'll review those new feaurs.

The Liberty Profile is shipped with WAS V8.5 but any given server instance is not part of a traditional WAS cell or node structure. Each Liberty Profile server instance is managed by way of its configuration file (`server.xml`) ... though it is possible to share configuration elements between many servers to make managing multiple servers easier.

This workshop has an entire section on the Liberty Profile where more details will be provided.

Applications

Overview of application development, packaging and deployment

Different Kinds of Applications

Here's a partial review of some common application "types" ...

Browser

Servlet / JSP

- Used to render user browser screens
- May interact with backend data
- May include JavaServer Faces or Struts (open source frameworks)

SOAP or RESTful

Web Service

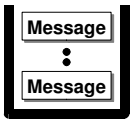
- Computer-to-Computer interaction
- Mobile devices
- Request type is HTTP

Servlet or Java Client

RMI/IIOP

EJB

- Often used as backend to Servlet/JSP
- Often used as business logic and interaction with backend data
- Often packaged with Servlet/JSP



MDB

- MDB = Message Drive Bean
- Listens for message arrival on queue
- Picks message up and triggers action of application (often calls other EJBs)

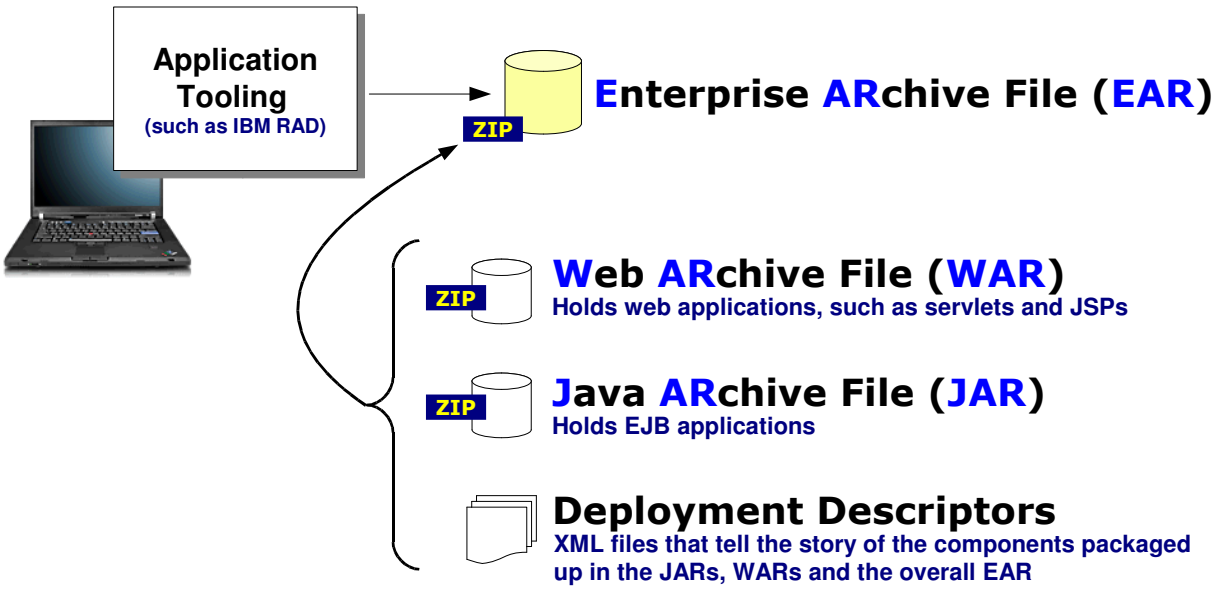
Packaging ...

Here we'll take a brief review of different kinds of applications you may encounter when using WAS z/OS (or WAS on any platform for that matter). This list of applications is not exhaustive ... you may well encounter other types. And you will almost certainly encounter applications that have a mixture of these elements included in the application being deployed. The purpose of this chart is really just to get concepts and terminology on the table.

- A very common application type is a servlet/JSP model. This is used when browsers are the interface for users accessing the application. JSP stands for "Java Server Page" and that provides a means of formatting dynamic web pages that contain some static content (logos, other graphics) as well as dynamic content (information specific to the user). Servlets are Java code that often provide the navigation logic between elements of the application and often do backend data access as well.
- Web services are becoming increasing common as well. Web services are intended for device-to-device communications, including the more recent mobile device access. The remote device communicates with either SOAP (Simple Object Access Protocol ... a standard for XML exchanges) or RESTful, which is based on commands included as part of the URL sent from the device. From a WAS perspective, web service requests are seen as HTTP requests, which is the same as used for browser access.
- Another application type is EJB, which stands for "Enterprise Java Bean." These are typically used for deeper business logic, and end-users access these EJBs usually through another component such as a servlet. Servlets and EJBs are very commonly packaged together and deployed a logical group into WAS.
- The final type mentioned on this chart is MDB, which stands for "Message Driven Bean." This is a way to have an application listen for the arrival of a message on a message queue (MQ or other messaging queueing function), and when the message arrives then pick it up and act on it.

Application Packaging and Deployment

The unit of deployment is an "EAR" file -- a zip format file -- that the DMGR takes in, determines requirements, then updates XML so appservers understand updates:



Deployment ...

WAS applications are comprised of not just one file, but dozens to hundreds of Java files. If you had to install each of those files individually it would be very time consuming and very prone to error.

Thankfully the Java EE standard defines a packaging format that makes things a little simpler. The basic unit for application installation is something called an "Enterprise Archive" file, or EAR for short. That is really just a ZIP-format file with a standard internal layout of folders and files. WAS knows how to crack open an EAR file and interrogate its contents as part of installation. EAR files are typically constructed by the application development tools used by the developers ... IBM Rational Application Developer (RAD) for example.

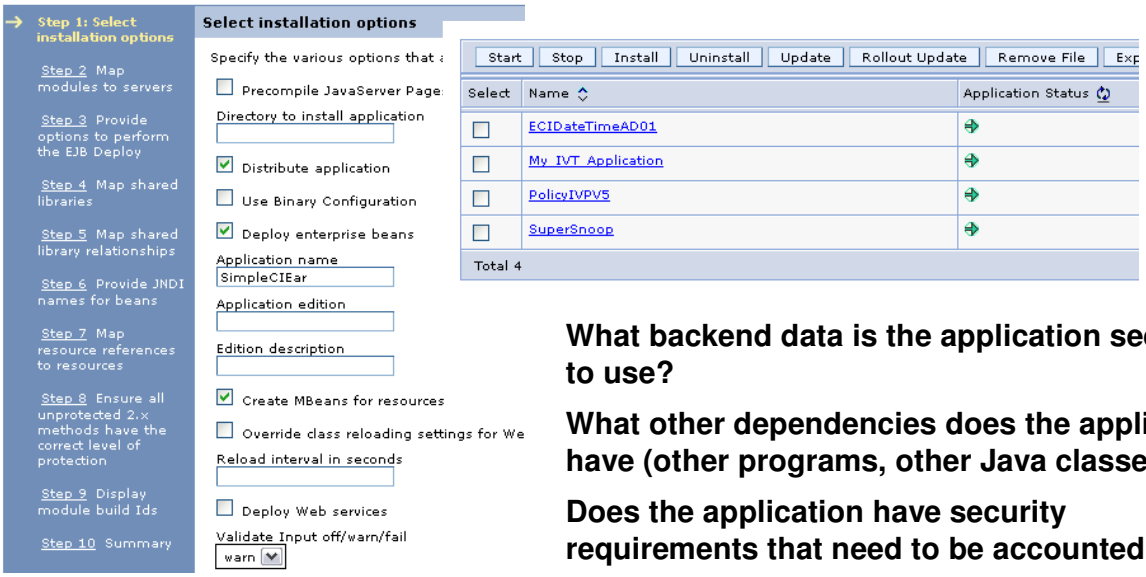
Think of an EAR file as a kind of "outer envelope" for application component packaging inside. There are two primary types of application components -- web components (servlets, JSPs) and enterrpise Java bean (EJB) components. Web components are packaged into a sub-ZIP called a "Web Archive" file, or WAR for short. EJB components are packaged into a "Java Archive," or JAR zip file.

In addition to the zip files there are XML files called "deployment descriptors" that "tell the story" of the application being installed. This is what WAS is looking for when you install an EAR ... it cracks open the EAR and finds the deployment descriptors, and from those it can tell what kind of application you're installing and what kinds of questions it's to ask of you to complete the installation.

EAR, WAR and JAR files have been around for a long time. They're not new to WAS z/OS V8. And they are platform-neutral ... an EAR file may be deployed to WAS z/OS or WAS Linux or WAS Windows.

Applications Deployment in WAS z/OS

Application deployment is the same on WAS z/OS as it is on other WAS platforms. Can be done through Admin Console or WSADMIN. Some things to consider:



What backend data is the application seeking to use?

What other dependencies does the application have (other programs, other Java classes)?

Does the application have security requirements that need to be accounted for?

In general it is best if application developers and WAS administrators communicate with each other so deployment is as successful as possible

Taking advantage of platform …

Applications are deployed through the Admin Console (or WSADMIN scripting). The process involves pointing to the EAR file to be deployed and then working through a series of steps where the Admin Console application as you (the human deployer) to resolve any required information needed to properly install the application.

How many steps of information input are required to deploy an application? It depends on the application, and it depends on what information is already encoded in the "deployment descriptors" packaged with the application.

The key point here is that an application may require things to exist in the WAS runtime environment for the application to be deployed correctly. These include data connections (which we'll cover in Unit 4 of this workshop), dependencies on other Java class files to be available, or security requirements such as EJBROLE settings.

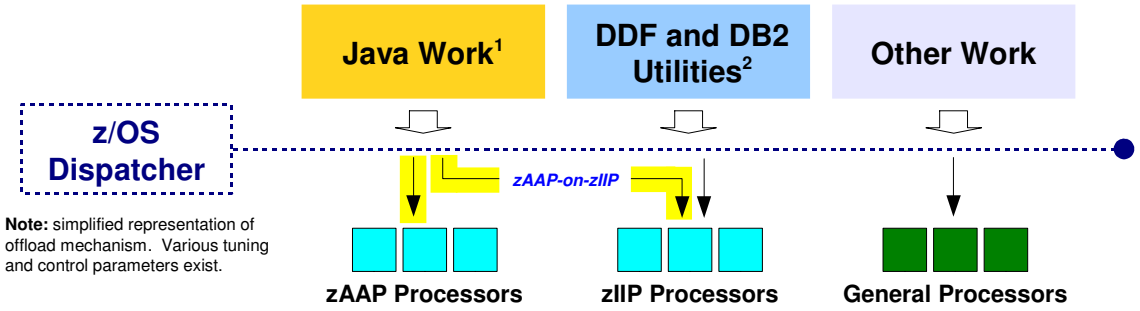
Application deployments should be done with proper communication between the developers and the deployers. It is not realistic to assume deployers can read the minds of the developers and know what is expected for any given application. A best practice is to developer procedures for documenting application requirements and passing those requirements along from developer to deployer.

Taking Advantage of z/OS

A review of the way WAS z/OS takes advantage of the platform

System z Specialty Engines

Specialty engines provide additional processing capacity with an attractive financial profile: lower acquisition cost, not counted towards software license charges:



zAAP - *System z Application Assist Processor*

Offload of Java and XML parsing work.

zIIP - *System z Integrated Information Processor*

Certain DB2 work and XML parsing services.

zAAP-on-zIIP

A means of more efficiently using specialty engines by defining only a pool of zIIP processors and allowing eligible zAAP work to run on the zIIPs[3].

IFL - *Integrated Facility for Linux*

For running z/VM and Linux. Does not apply to z/OS, but plays strong role in Linux for System z

Note 1 -- See http://www.ibm.com/systems/z/hardware/features/zaap/
Note 2 -- Plus other work, see http://www.ibm.com/systems/z/hardware/features/ziip/
Note 3 -- EC12 planned to be the last system that supports zAAP; after that, zAAP-on-zIIP will be the offload mechanism

We start this discussion with a high-level explanation of System z "specialty engines" for those who may not have a background in the topic. The ability to offload Java work to specialty engines is one of the key motivators to using Java on System z.

Specialty engines provide a means of dispatching work running on z/OS to engines other than the general processor engines. That provides two key benefits:

- It allows work that must run on general processors (such as CICS, or COBOL batch) to have capacity on those engines by not using GP cycles for Java, and
- It allows Java work to run on engines that do not count towards software licensing charges.

The net effect is that adding Java work to a z/OS system does not necessarily imply having to acquire addition GP engines, which then implies potential increases in software licensing charges. It is possible that Java work can have a cost-neutral effect (aside from the acquisition cost of the specialty engines, which are priced substantially below GP engines).

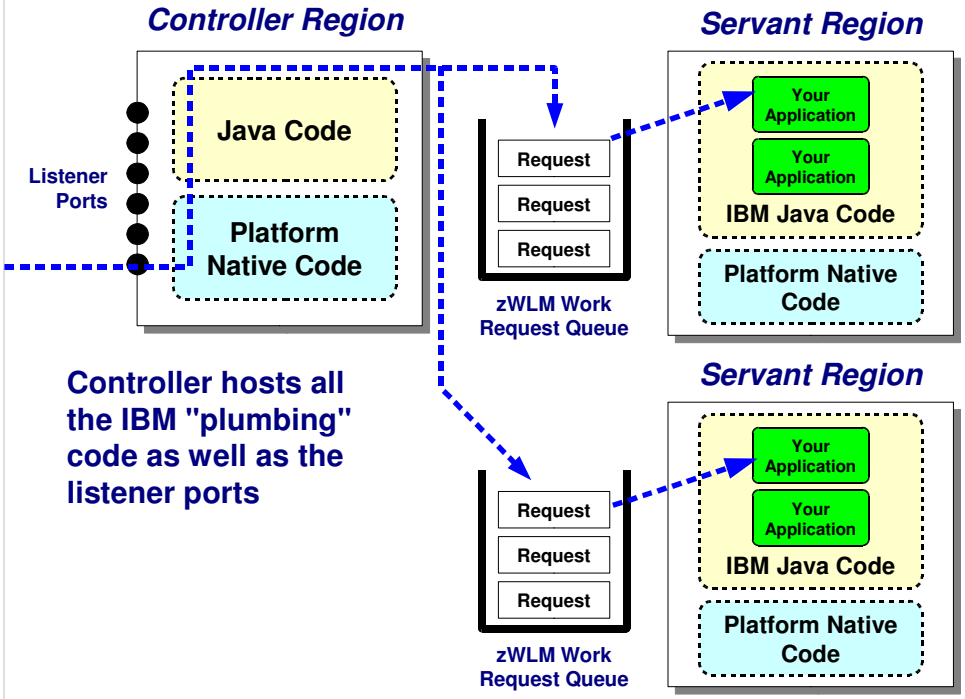
Note: the dispatching of Java work to available specialty engines is transparent to the Java applications. The decision where to dispatch is done at a level lower on the system ... down at the interface between the JVM and the z/OS dispatching code.

There are several flavors of specialty engines and those are discussed on the chart above. With respect to Java the key is zAAP, or zAAP-on-zIIP. (The EC12 machine is the last to support zAAP alone; going forward it will be zAAP-on-zIIP, which is a means of consolidating non-GP work on a pool of specialty engines rather than having to separately allocate zAAP and zIIP engines. In short, zAAP-on-zIIP is a good thing.)

In summary, the existence of specialty engines on z/OS is a key enabler for Java work. It is a factor often cited by customers as a factor that plays a role in their decision to host Java work on the mainframe.

WAS z/OS and the "Multi-JVM" Design

We have a whole section on exploiting this feature. For now, focus on the essentials:



TechDocs WP101740 Clusters ...

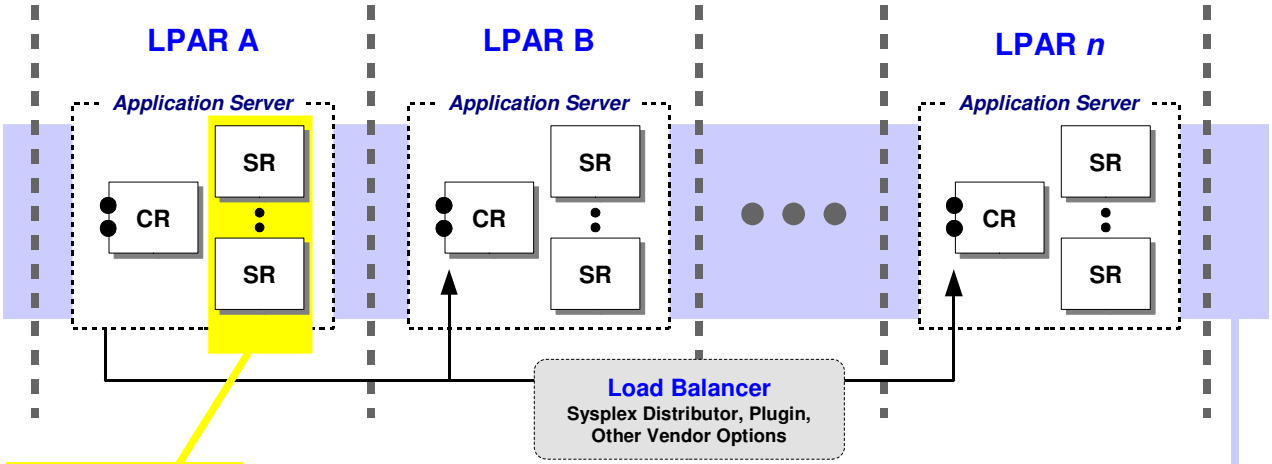
One of the most striking differences between WAS on z/OS vs. WAS on other platforms is the "multi-JVM" model of the application server. On other platforms the "application server" is a single JVM while on z/OS the application server is at least two JVMs and potentially more.

You may be familiar with this already. If so then later on we have a whole unit on this model that will explain in more detail what takes place between the controller region and the servant region, and explore why certain things behave the way they do. If you're not familiar with this model, then a brief introduction is in order:

- The "Controller Region" (or "CR" for short) runs a JVM that is used for IBM plumbing code. Your applications do not run here. This is used to host the listener ports, to take in the requests and classify them, and to queue the work to the servant region where it runs.
- The "Servant Region" (or "SR" for short) runs a JVM where your applications execute. For any given application server instance you will have one CR and between 1 and *n* servant regions. If you have multiple servant regions then your application binaries execute in each of the servants. Right there you can see that the multi-JVM model provides a kind of "vertical cluster" behind the CR to provide a degree of availability.
- The number of servant regions that execute is configurable by you. You may elect to have only one SR for a server, or you may elect to have four started when the server starts, or you may elect to allow z/OS WLM to dynamically start additional servants if it sees a single servant is not meeting goals. Again, the unit coming up will go into details on how all that is configured and managed.
- In between the CR and the SRs sit z/OS WLM work queues where work requests are very temporarily "parked" before being dispatched to the servant regions for execution. This provides a kind of short term "shock absorber" between the CR and SR. It also explains why with WAS z/OS it is not necessary to configure hundreds or thousands of threads to handle temporary spikes in requests. On WAS z/OS those requests temporarily in excess of ability to process are "parked" on the WLM work queues rather than "parked" on threads. Final point -- the SR by definition can't be overrun with work ... it will take only what it can process. If the available servants fall behind then the WLM work queues expland. This is a pull model, not a push ... the SR can't be overrun because it pulls ... the CR does not push.

Clusters - "Inner" and "Outer"

With WAS z/OS we have two levels of clustering for availability:



"Inner" Cluster

- Multiple SRs behind a CR
- Each SR physically separate JVM
- App binaries in each JVM
- Each SR has own worker thread pool
- WLM will restart failed SR
- WLM will distribute work (Unit 3)
- Stateful replication possible

"Outer" Cluster

- Multiple appservers across LPARs
- WebSphere cluster common across platforms
- App binaries in each appserver's SRs
- Stateful replication possible
- Many options for front-end work distribution

Other examples …

With WebSphere Application Server for z/OS we have the opportunity to form up two levels of clustering for application availability.

The first level is the "inner" cluster, which is formed by configuring multiple servant regions (SRs) behind a controller (CR). A CR and its associated SRs comprise a logical "application server." Each SR is a physically separate JVM with all the application binaries available for applications deployed to the server. Each SR has its own worker thread pool, which means if you have 10 worker threads configured for the server, the first servant region has 10 and the second has 10 more. WLM will automatically restart failed servant regions to protect the configured minimum SR number, and WLM will distribute work between the SRs. We cover all this in much more detail in Unit 3 of this workshop. Finally, stateful data may be replicated between the SRs so that a lost SR does not imply a client having to log back in and re-enter whatever data they'd entered up to the point of SR failure.

The second level is the "outer" cluster, which is what WAS on every platform has the ability to do. This is duplicated application servers across operating system instances. In a z/OS environment that's typically done by duplicating servers across logical partitions (LPARs). Each application server will have the application binaries loaded for applications deployed to the cluster. As with the inner cluster, stateful data replication is possible. Work distribution to a cluster may be accomplished in a variety of ways, including using z/OS Sysplex Distributor if you wish.

The objective of application duplication is availability and scalability. And while duplicating application instances is not the total HA story (which has many layers to it beyond simply duplicating applications), the *starting point* is duplication of app instances. WAS z/OS provides two levels of that -- inner and outer as shown.

Note: the terms "inner cluster" and "outer cluster" are informal terms we've adopted for this. You will not find references in the official documentation for those phrases. But they are useful because they differentiate the two levels of clustering available with WAS z/OS.

WLM, SAF, SMF, MODIFY, Cross-Memory

Other points of platform exploitation are those summarized here:

WLM

z/OS Workload Manager

- Controller / Servant structure as discussed on previous chart
- Request classification for separate service classes and reporting classes

SAF

z/OS Security Access Facility

- Sysplex-aware security definition repository and resource access control
- Userids and passwords, SSL certificates, EJBROLE definitions
- Security workshop covers WAS z/OS security in detail (ask for details if interested)

SMF

z/OS System Management Facilities

- SMF 120.9 record to record detailed information about request activity
- Useful for analysis and chargeback
- See WP102205 at ibm.com/support/techdocs for guide to SMF Techdocs

MODIFY

z/OS MODIFY interface

- Allows dynamic operations against WAS z/OS servers
- Long list of actions to display and act up on server operational behavior

Cross Memory

z/OS Cross-Memory Exchange

- DB2 Type 2 connector, CICS EXCI, MQ BINDINGS, WOLA
- Low latency, better security

Wrap up ...

This chart provides a quick summary of the other ways in which WAS z/OS takes advantage of the platform. This is a fairly high level summary ... a deeper review of the "Why WAS z/OS" question can be found at the Techdoc by that same name:

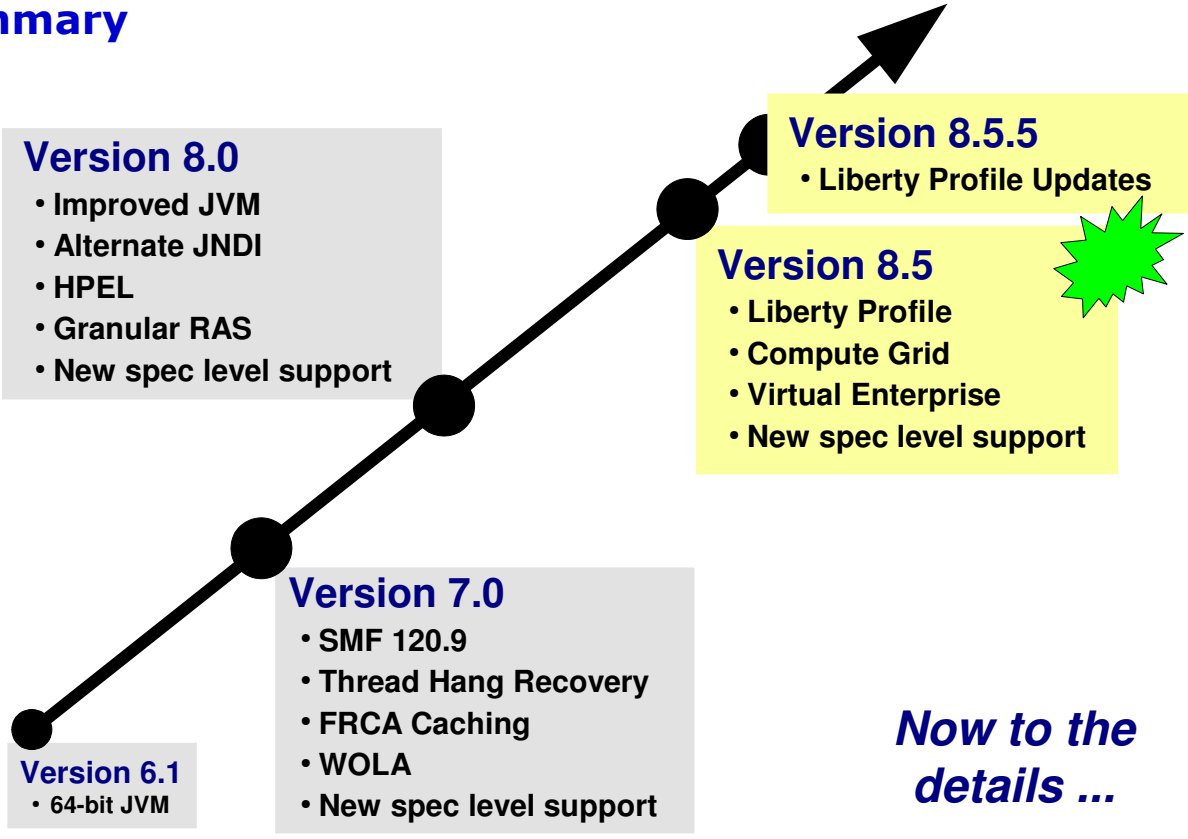
http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101532

The green stars indicate topics we'll cover in more detail in this workshop.

Wrap-Up

Final thoughts before getting to the rest of the workshop

Summary

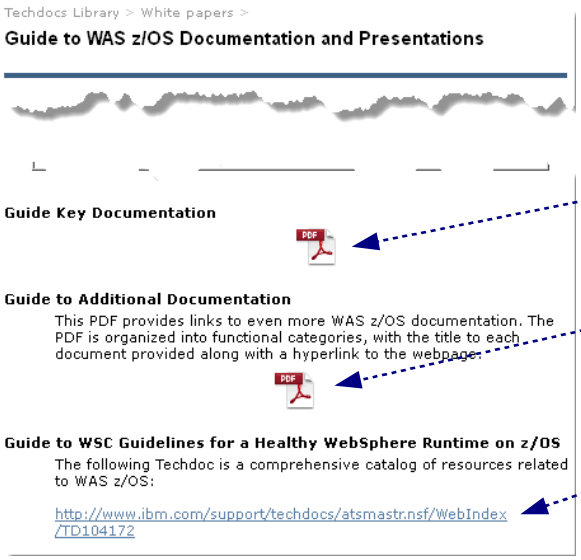


This chart provides a very high-level summary of where we've been and what V8.5 brings to the table as part of that history. Now it's time to get to some details.

Techdocs

We've published a great deal of useful information out on the Techdocs site. So many that we decided to publish a "guide" to all the documents ... WP102205

ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102205



PDF with hiperlinks to the key documents for WAS z/OS

PDF with hiperlinks to important documents not listed in the "key documents" PDF

John Hutchinson's "Healthy Runtime" information

The labs ...

We have a lot of Techdocs out there. To help you understand what documents are available, we produced a guide to the documents ... a kind of "table of contents" to the key documents we've published.

Few Notes About the Labs

Slow and steady ... lots of information, so trying to rush usually results in overlooking things

MVS and ISPF usage hints in the back

Cut-and-paste command text file on desktop

WBSR85 Lab Commands for Cut-and-Paste .txt

```
**************************************
* UNIT TWO LAB - ADMINISTRATIVE MODEL *
**************************************
S Z9DCR,JOBNAME=Z9DMGR,ENV=Z9CELL.Z9DMNODE.Z9DMGR

S Z9ACRA,JOBNAME=Z9AGNTA,ENV=Z9CELL.Z9NODEA.Z9AGNTA

http://wg31.washington.ibm.com:10005/ibm/console

df | grep /wasv85config/z9cell

df | grep SBBOHFS

cd /wasv85config/z9cell/z9dmnode/DeploymentManager/profiles/default/bin
```


A few notes about the upcomings hands-on labs.

End of Unit