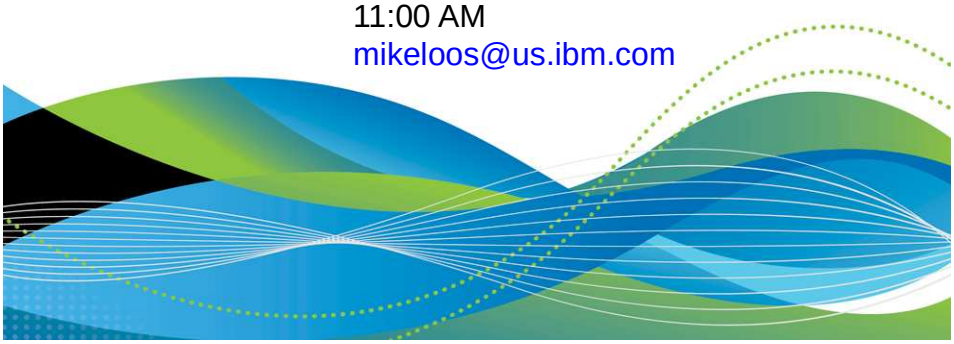
#SHAREorg

SHARE
Technology • Connections • Results

WebSphere Application Server on z/OS Back to Basics

Mike Loos
IBM
Session 10580
Monday, March 12, 2012
11:00 AM
mikeloos@us.ibm.com



WebSphere Application Server on z/OS

Session	Day	Time	Room	Title	Speaker
10560	Monday	9:30	International Ballroom F	Version 8 – Overview and Update	David Follis
10580	Monday	11:00	Cottonwood A/B	Back to Basics	Mike Loos
10633	Wednesday	1:30	International Ballroom C	Installation Manager – The Cross Platform Installer for WAS	Mike Loos
10561	Wednesday	3:00	Cottonwood A/B	Version 8 – New z/OS Exploitation Features	David Follis
10562	Thursday	11:00	Cottonwood A/B	Batch Update	John Hutchinson
10581	Thursday	1:30	Cottonwood A/B	Getting Started with Version 8 – Part Zero!	Mike Loos
10518	Thursday	6:00	Cottonwood A/B	Potpourri	Anybody
10516	Friday	8:00	Dogwood B	Level 2 Update	Mike Stephen
10563	Friday	9:30	Pine	Hands on Lab	Mike Stephen, David Follis, Ken Irwin

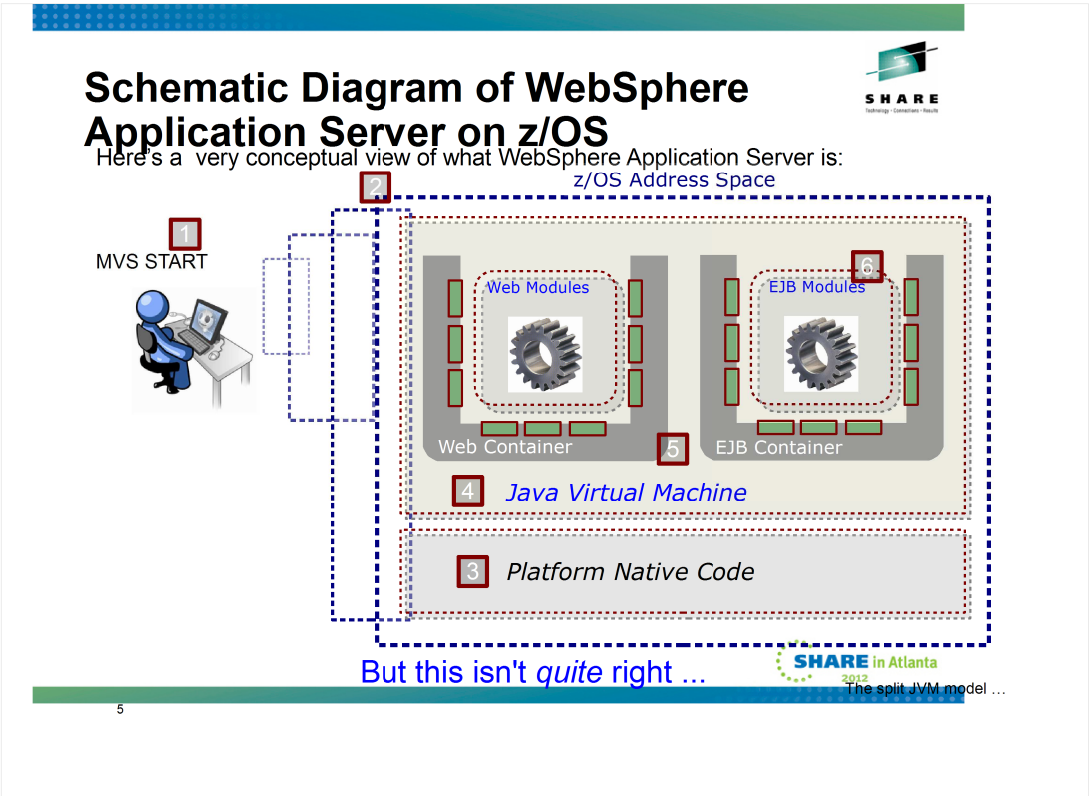
Starting out...

- Introduction.
- WebSphere on z/OS Overview
- Overview of the Installation Process.
- ”Get Ready” stuff...
- Some Setup Info…
- Install the WebSphere code…
- Using the WCT to Configure WebSphere...
- Adding an Additional Node.
- Creating a Server.
- Creating to a Cluster.
- Some Useful Stuff (Config tweaking).
  - Setting some variables.
  - Setting console preferences.
  - Use of the RMI connector.
- Etc...
- Q & A?

This presentation is the accumulation of experiences installing WebSphere on z/OS at customer locations across the US in combination with those acquired working with the folks from the WSC, with input from WebSphere on z/OS level 2 support.

This overview is extracted from the WBSR7 wildfire class, the full content of which is available on the techdocs website at the following url:

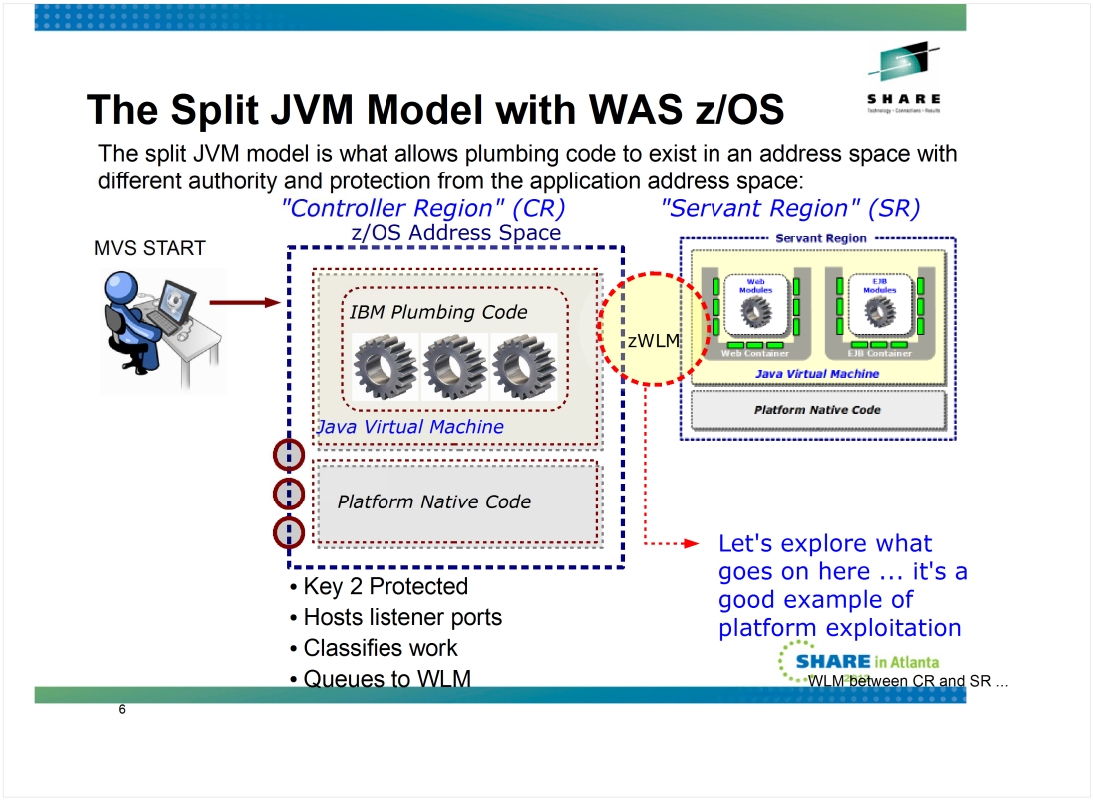
http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS3422



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. A WAS z/OS application server is started with a standard MVS START command.
2. On z/OS that implies an address space, which is *roughly* analogous to a UNIX process.
3. 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.
4. Once the foundation is set, then the Java environment is started up.
5. With the JVM in place, WAS can now load the various Java class files that implement the open standard APIs and the function behind it. They are organized into two "containers" -- one for web modules and one for EJB modules.
6. Finally your application modules are loaded in and started.

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.

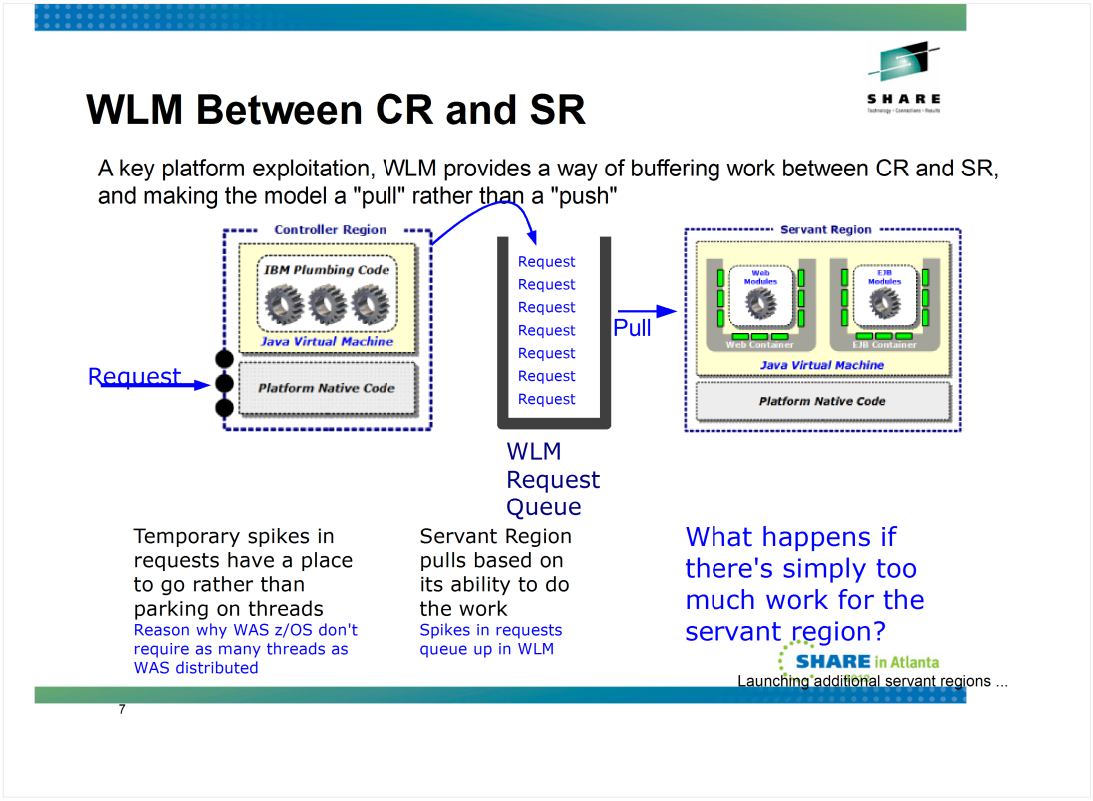


WAS z/OS is different from WAS on other platforms in that each application server operates with a "split JVM" model -- a "controller region" and a "servant region." The CR provides the initial handling of requests and does workload classification. The SR is where the applications run and where the "container" structures are implemented.

Note: for those with some familiarity with WAS z/OS ... yes, multiple SRs is possible. We'll cover that soon.

Between the CR and the SR sits the z/OS Workload Manager (zWLM, or just WLM). We put the "z" on the front of that because WAS on the other platforms has something termed "workload manager" but it is not the same thing as z/OS WLM.

WLM is what starts the SR after the CR initializes. WLM is also what maintains work request queues, which is what we'll explore next.

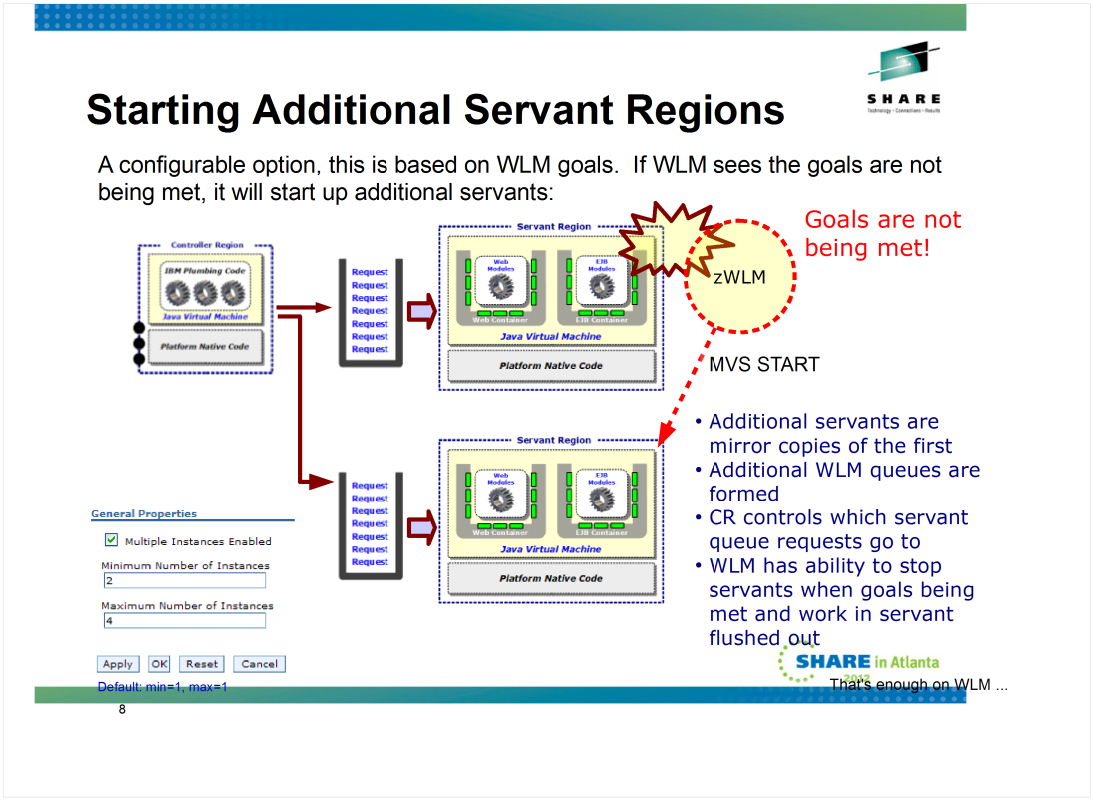


Sitting between the CR and the SR is a WLM work request queue where work is placed prior to the SR taking the work to service the request. This is what provides the ability to take a spike in requests and not overload the system. The CR will take the work in and park it on the WLM queue. These are not execution threads, these are very low overhead memory structures.

The SR takes work as it's able. It won't take more than it can because it's just a matter of a thread freeing up and taking the next unit of work.

Note: there's actually quite a bit more sophistication in the middle of all this. We're simplifying things here to make key points.

What happens if the inbound work is too much for the servant to service? If you have it configured to allow it, the CR can ask WLM to start up additional servant regions. That's next.



WLM is a very sophisticated system resource monitoring and allocation control mechanism. We can't go into all the details of WLM here, but suffice to say that WLM watches the state of all activity and compares against the work goals you've defined.

Imagine a servant region is taking work as fast as it can, but WLM still sees that the defined goals are not being met. If you've allowed it (it's a configurable option), WLM will start additional servants. Doing this results in additional WLM work queues being created and the CR placing work requests on the queues of the servants.

Note: there is considerable sophistication in the middle of this ... far more than we can cover on one chart here. We'll point you to a technical document that explains all this in much closer detail.

When the workload surge cools off, WLM has the ability to stop allocating work to the excess servants and allow work there to flush out. Then stop that servant.

Techdoc WP101470

Written by Dave Follis, an architect of WAS z/OS, it details the inner workings of the CR/SR/WLM interactions and reveals all the capabilities it provides

Document Author: Don Bagwell
Additional Author(s): David Follis
Document ID: **WP101740**
Doc. Organization: Advanced Technical Skills
Document Revised: 09/01/2010
Product(s) covered: WebSphere Application Server for z/OS; WLM; z/OS

Abstract: This paper explains in some detail the way in which WAS z/OS makes use of the z/OS Workload Manager (WLM). Most pictures of the WAS z/OS controller / servant region show a single WLM queue, but in fact there may be several. This paper explains this and helps show the power and value of the WAS z/OS platform.

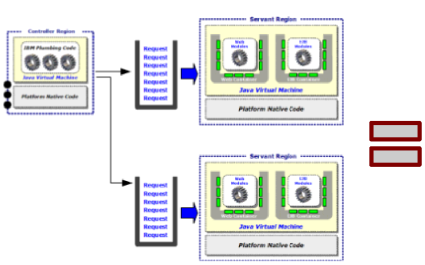
We'll leave this topic and turn our attention to the overall topology

ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101740

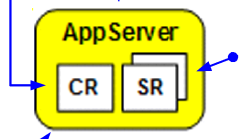
If the CR / SR interaction with WLM interests you, by all means pull Dave Follis' Techdoc on the subject.

Simplifying the Picture of the AppServer

Going forward, we'll use a simplified icon to represent the CR / SR structure we talked about earlier:



Controller Region
One address space



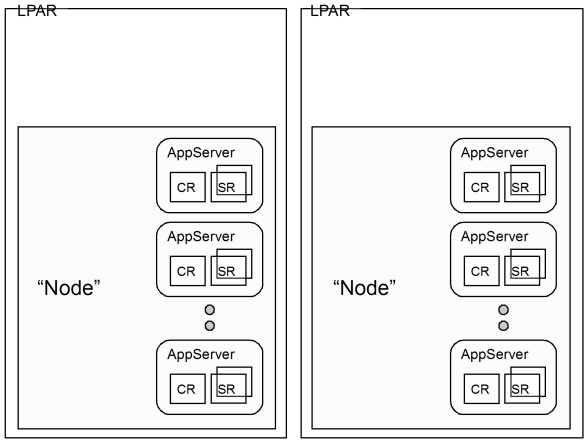
Servant Region(s)
One or more address spaces, depending on how you configure it

AppServer
A logical grouping of the CR and SR.
To WAS, this grouping is the "AppServer"

Going forward from here we'll use a little stylized icon to represent the CR / SR structure. The icon we'll use is shown on the chart. The square blocks represent z/OS address spaces; the curved outer box represents the logical collection which WAS itself views as the "appserver."

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)

Deployment Manager …

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

11

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.

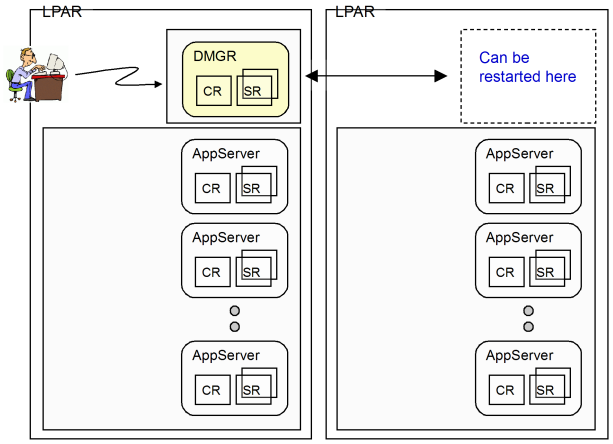
There is no limit to how many servers you can have in a node. It's really a question of resources on the LPAR to support the address spaces.

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.

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.

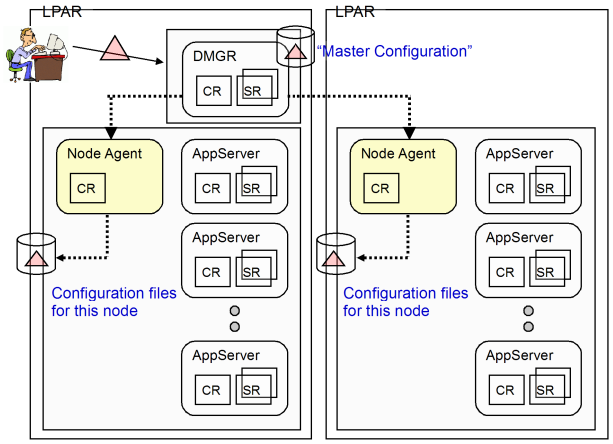
Note: this is a good time to point out that the DMGR is **not** required for the steady-state operation of the other servers in the configuration, including your applications. The DMGR can be down and your applications, running in application servers, can happily continue on.

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.**

SHARE in Atlanta 2012

The cell …

13

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:

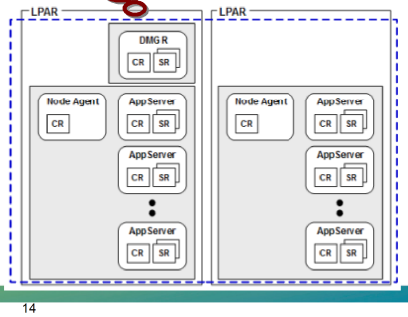
- You make changes to the configuration through the Administrative Console, which runs on the DMGR. The DMGR updates its "master configuration" -- which is maintains for the whole "cell" (which we've not yet explained, but think of it as everything managed by the DMGR).
- The DMGR then taps the Node 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.

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.

My XML tells me I manage two nodes. Therefore, that is the extent of my "cell"



A logical construct ... not a started task

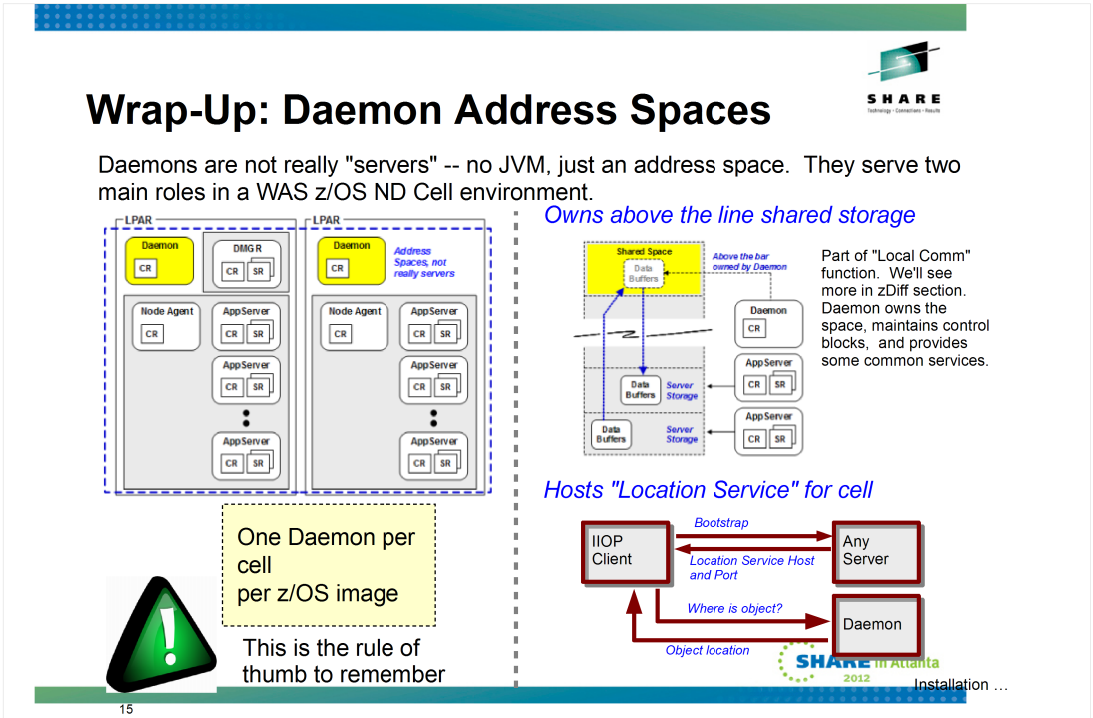
Is important because it is often used as the means of isolating at the security and purpose level -- test, QA, production

The cell is automatically created and automatically expanded as more nodes are built and added to it.

- Multiple cells allowed ... per LPAR and per Sysplex
- Cells *can* span to non-z/OS platforms. That introduces a bit of complexity, but it is possible

The "cell" 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.



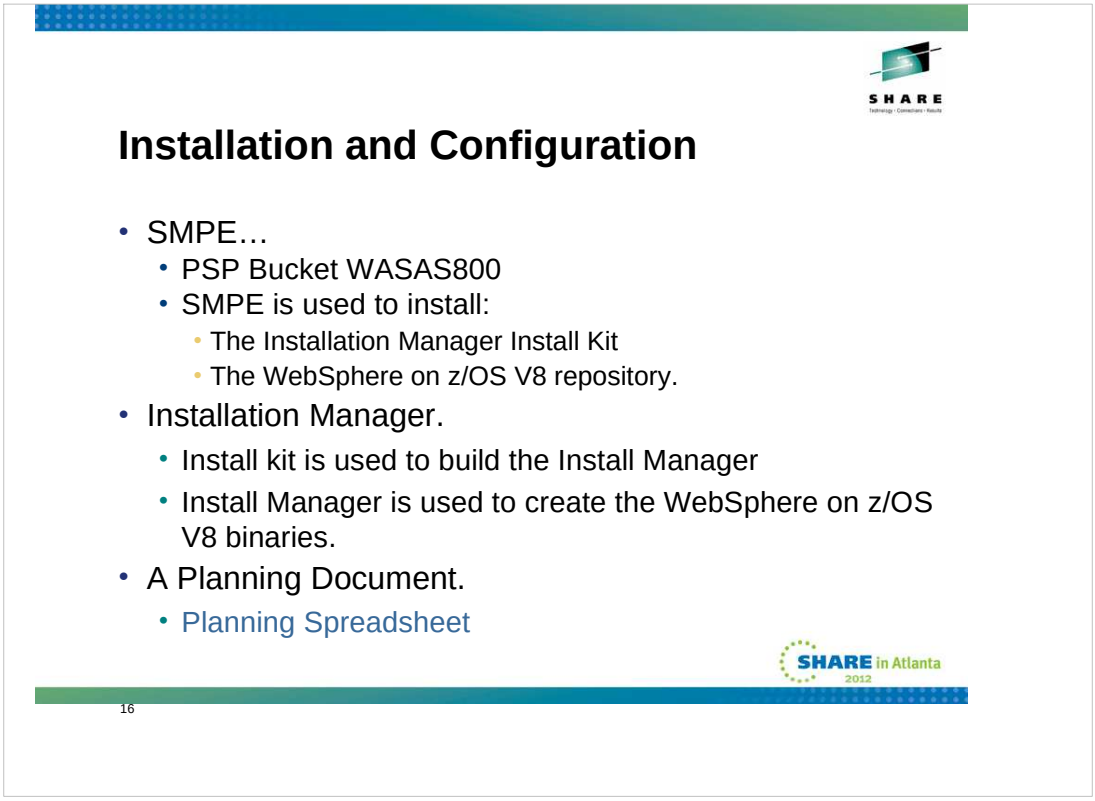
The Daemon server is something unique to WebSphere z/OS. It consists of a single controller region. It has no JVM and no Java code at all. It's not really a "server" like other WAS servers ... it's really more just an address space. It's not associated with nodes, it's really more a cell-level thing. But those are nuances you can, for now, overlook.

I's purpose in life is two-fold:

- It's what owns shared space above that line that's used for "local comm" -- cross memory data buffer exchanges for inter-server IIOP communications in a cell on the same LPAR. This becomes important because the WOLA function (we'll cover this in the zDiff section) is based on this. That ownership of the shared space is the reason why when a Daemon is stopped all the servers for that cell on that LPAR also stop. Lose the Daemon's shared space control and you lose the local comm.
- It provides the location name service so external clients seeking objects within the WebSphere cell can locate and bind. This is for RMI/IIOP requests coming in from EJB clients outside the WebSphere cell. (Note, a cell comprised of z/OS and distributed boxes -- which is possible -- is still one cell, and in that case an EJB client is operating within the cell. What we're referring to here with the Daemons is the case where you have a server box, unrelated to this cell, with an EJB client looking to connect and bind to an EJB in this cell. Then the location name service hosted in the Daemon servers take over.

Daemons are created at the time the node is created. There are some complex subtleties we'll explore later, but in general their creation is done when you run the jobs to create the cell. And while they can be started manually, in general they are started by WebSphere when the first server for the cell on that z/OS image starts.

The key here really is the yellow box "rule of thumb." Remember that and you'll be fine for this workshop.



Before you start it is important to pull the appropriate PSP bucket for your level of WebSphere, and of course, to comply with all of the recommendations that it contains.

The general flow is that a new component on z/OS, the Install Manager is used to create the WebSphere binaries. This starts out by the installation (either SMPE install or download of a zip file and expansion into a file system) of the Installation Manager Install Kit. The Install Kit is then used to create an Install Manager. The Install Manager is then used to create the WAS on z/OS binaries, using a WAS on z/OS repository that is initially installed using SMPE.

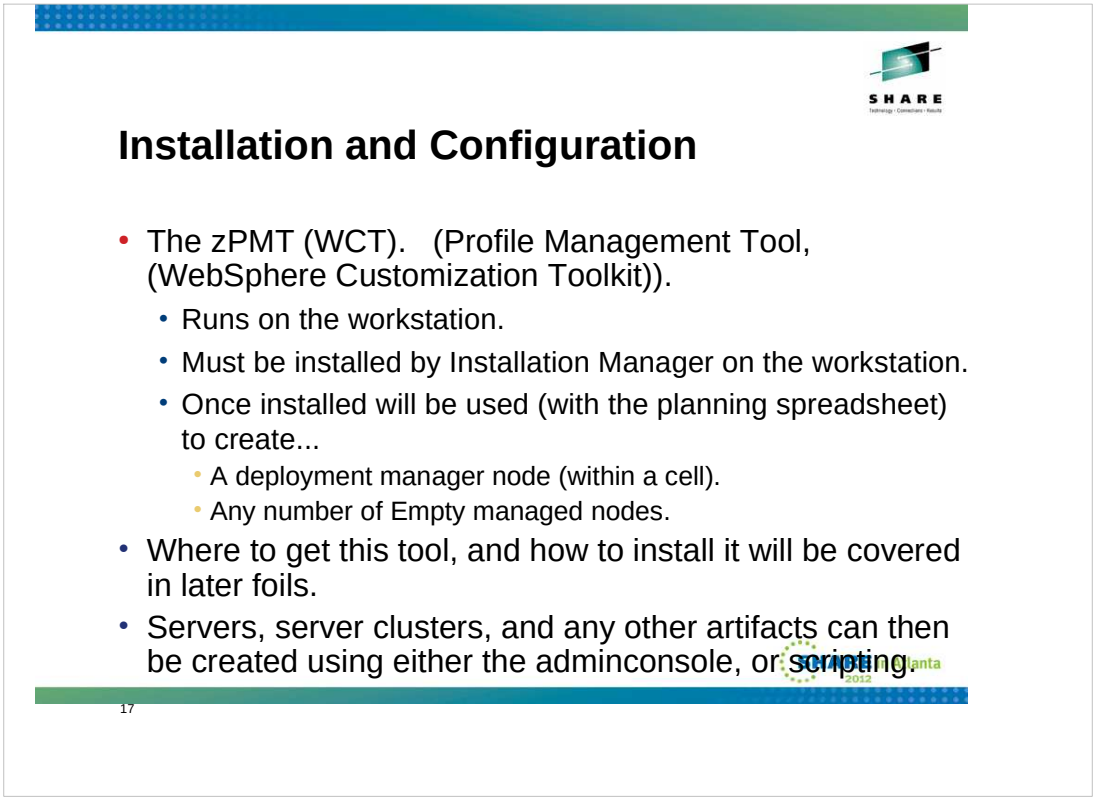
Once that much is complete, you may proceed with configuration. Configuration is unchanged from the previous release. The general flow of the construction of an ND (Network Deployment) Cell of WebSphere on z/OS is to build the Deployment Manager cell and server, build an Empty Managed Node or nodes, then add servers, clusters, etc., as necessary.

A good planning document is extremely important. An excellent worksheet may be obtained from the following link:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS4686

The sample used here is filled out for a cell which will be referred to as the S8 cell.

The zPMT, is now part of the WebSphere Configuration Toolkit (WCT), is used to build the JCL and data files necessary to create the configuration.

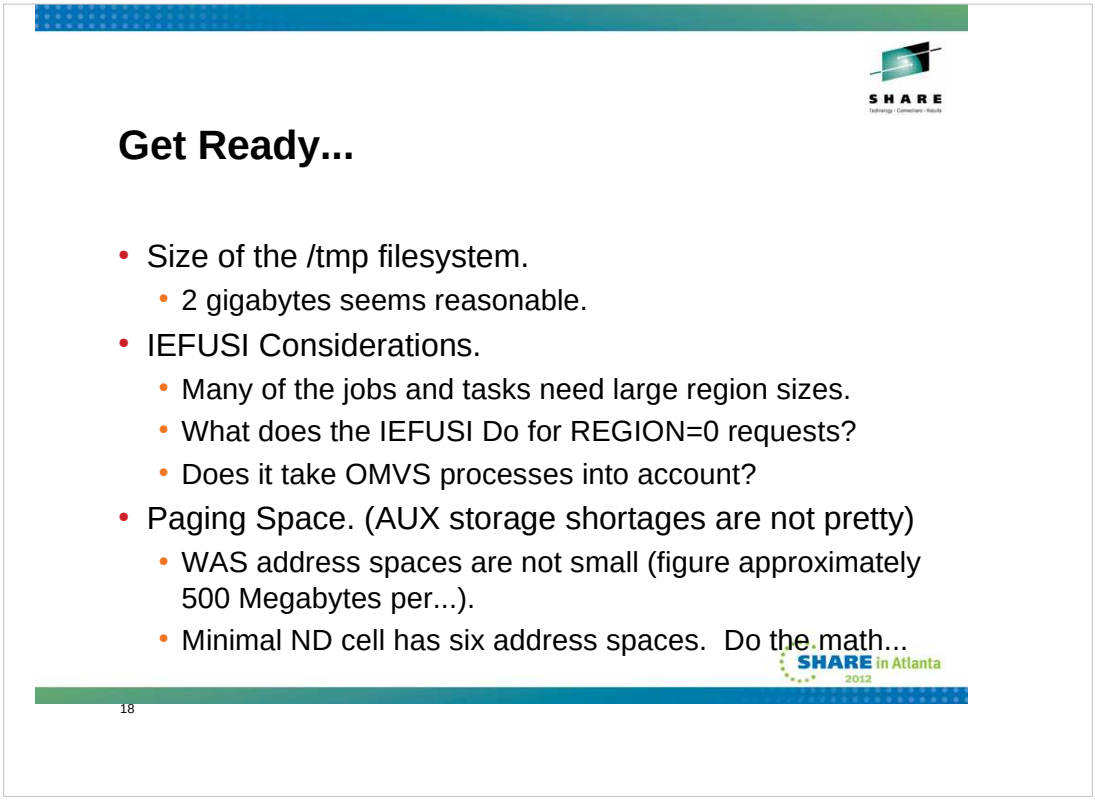


The zPMT (WCT) which runs on the workstation must be installed by the Installation Manager on the workstation (which you may also need to install).

Once the WCT has been installed, it will be used (in conjunction with the planning spreadsheet) to create:

1. A Deployment Manager (includes the dmgr server, dmgr node, and the cell).
2. Any number of Empty Managed Nodes.

Servers, server clusters, and any other artifacts can then be created using either the adminconsole or scripting.

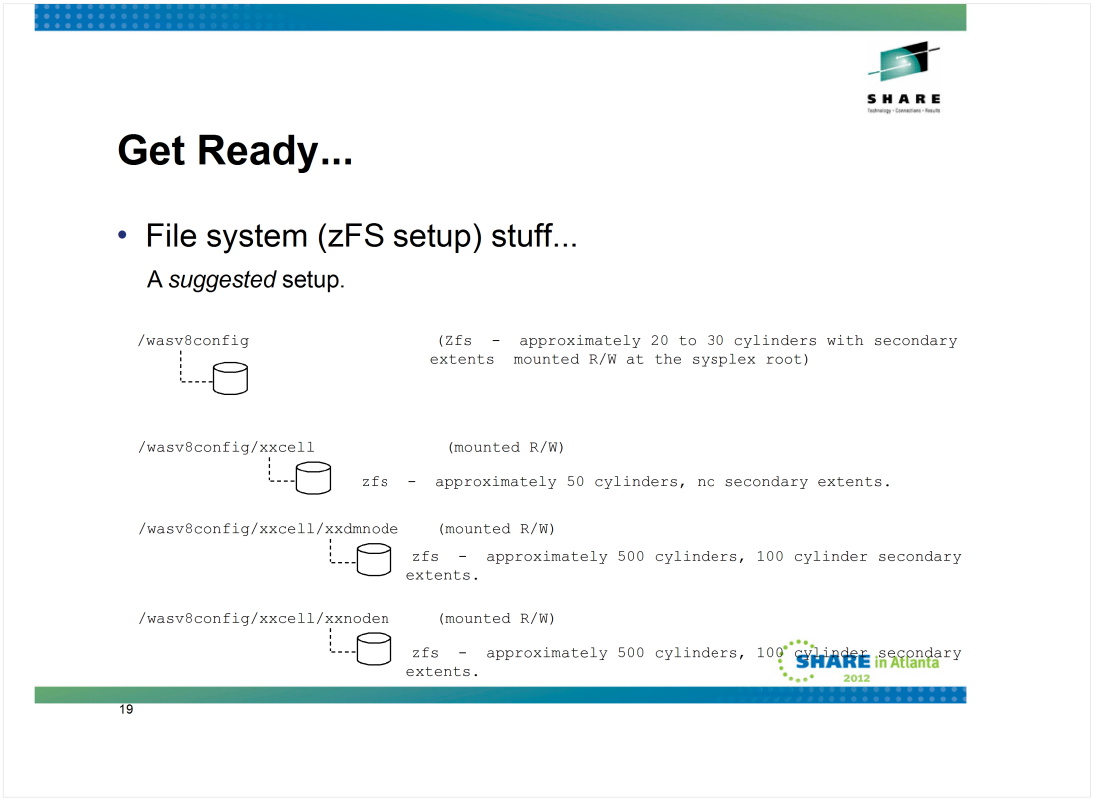


Some things to think about before you start doing anything with WebSphere on z/OS, or for that matter the Installation Manager, are:

1. Check and correct if necessary the size of the /tmp filesystem. The default is about 4 megabytes and is way too small. I'd suggest that about two gigabytes is a reasonable starting point.

2.If you have an IEFUSI exit active (other than the default one delivered with z/OS), you'll need to make sure that it accounts for your need for large region sizes. Also it should take OMVS processes into account by effectively ignoring them.

3.You may need to alter the amount of paging space available. Even if you are not paging, there needs to be enough auxiliary storage available to support the region sizes you are requesting. The WAS address spaces are large, roughly half a gigabyte per address space. A minimal ND cell has six address spaces, so that means an additional 3390-3 worth of paging space. You DO NOT want to run out!

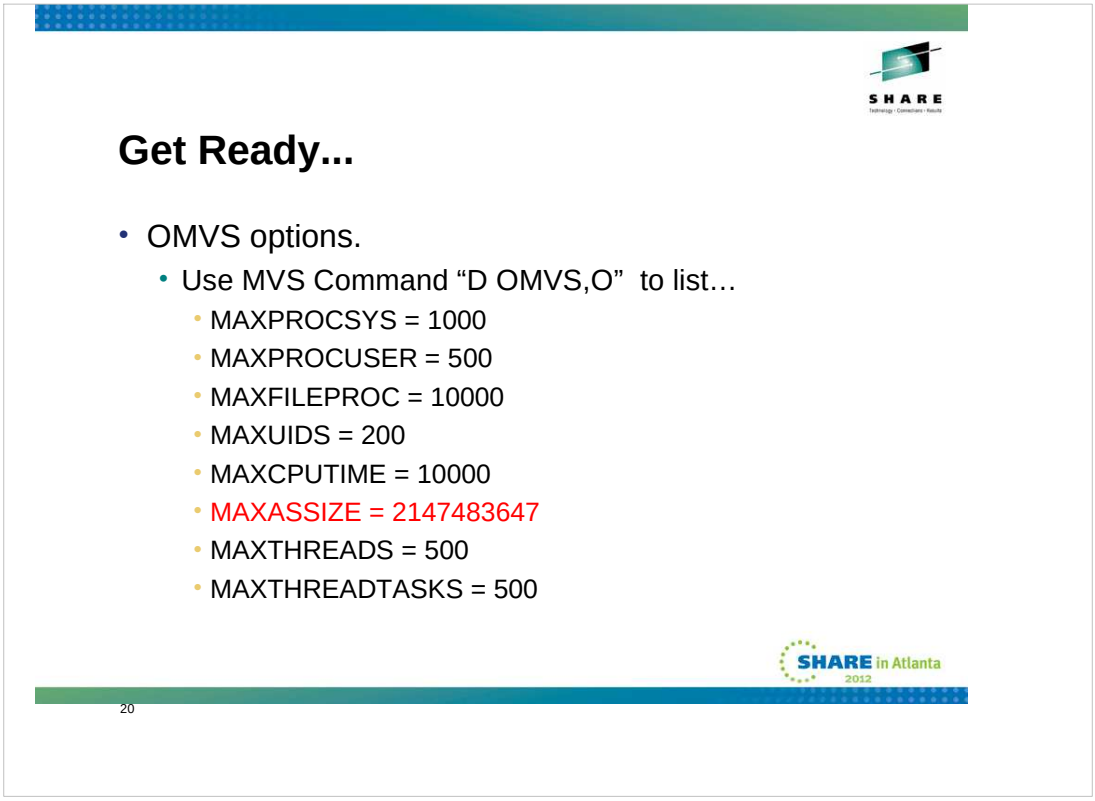


One of the first things that should be considered when creating a new WebSphere on z/OS configuration is the underlying file system(s).

The first file system to consider is what we'll refer to as the WebSphere root. This is basically a filesystem to hold other mountpoints so as to keep us out of the root, always a good thing. A good starting point is to make this file system 20 to 30 cylinders with secondary extents allowed and mount it read/write, usually in the sysplex root.

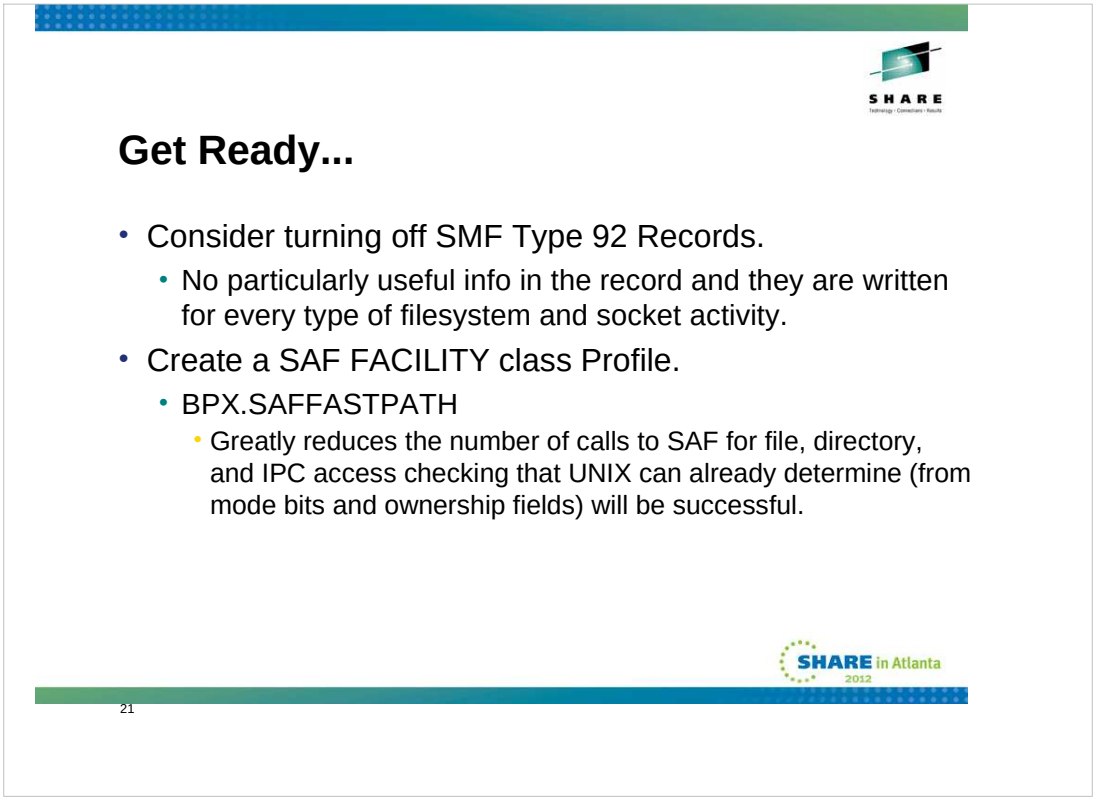
Next up is what I'll refer to as the “cell” root. There should be one of these for each cell, and it should be about 50 cylinders with no secondary extents, mounted read/write. The configuration file systems are mounted within this filesystem, as well as all of the userid “home” directories for the cell. An advantage to this is that, by default, some java dumps end up defaulting their location to the userid's home directory. Having them within this filesystem with no secondary extents should allow you to capture a couple of them without filling up a lot of space, and hopefully correcting the problem.

Last are the actual node configuration file systems.



Make sure that your OMVS options in BPXPRMxx are realistic for actually using USS for something other than TCPIP. The above values are basically minimums.

Of particular importance, is the value for MAXASSIZE with is basically set to 2 gigabytes minus 1. This essentially removes memory restrictions on OMVS processes, assuming you aren't doing something else with an exit (IEFUSI).



SMF type 92 records are basically written for any and all file system activity, as well as socket activity. If you have them turned on they may quickly become the prevalent type of record in your SMF files.

There is very seldom a need for the information they provide, so to save yourself the performance hit of collecting them and throwing them away (or storing them and never looking at them), just turn them off.

The BPX.SAFFASTPATH is very similar to the SMF 92 record hint. Instead of the type 92 record, this can cause an excess of RACF audit records.

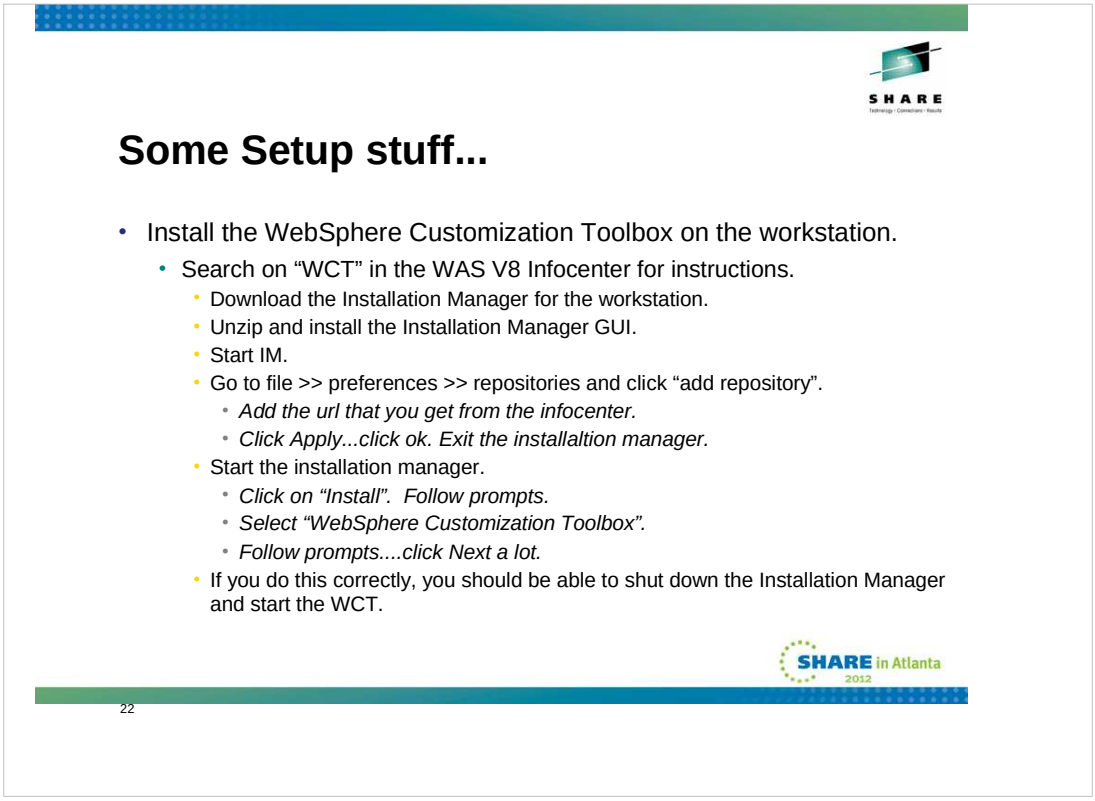
All you have to do to implement this is to define the RACF profile:

```
RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE) OWNER(SYS1)
```

and either IPL, restart OMVS, or cause it to refresh its self by issuing the following command:

```
SET OMVS=(XX)
```

where xx represents an empty BPXPRMxx member.

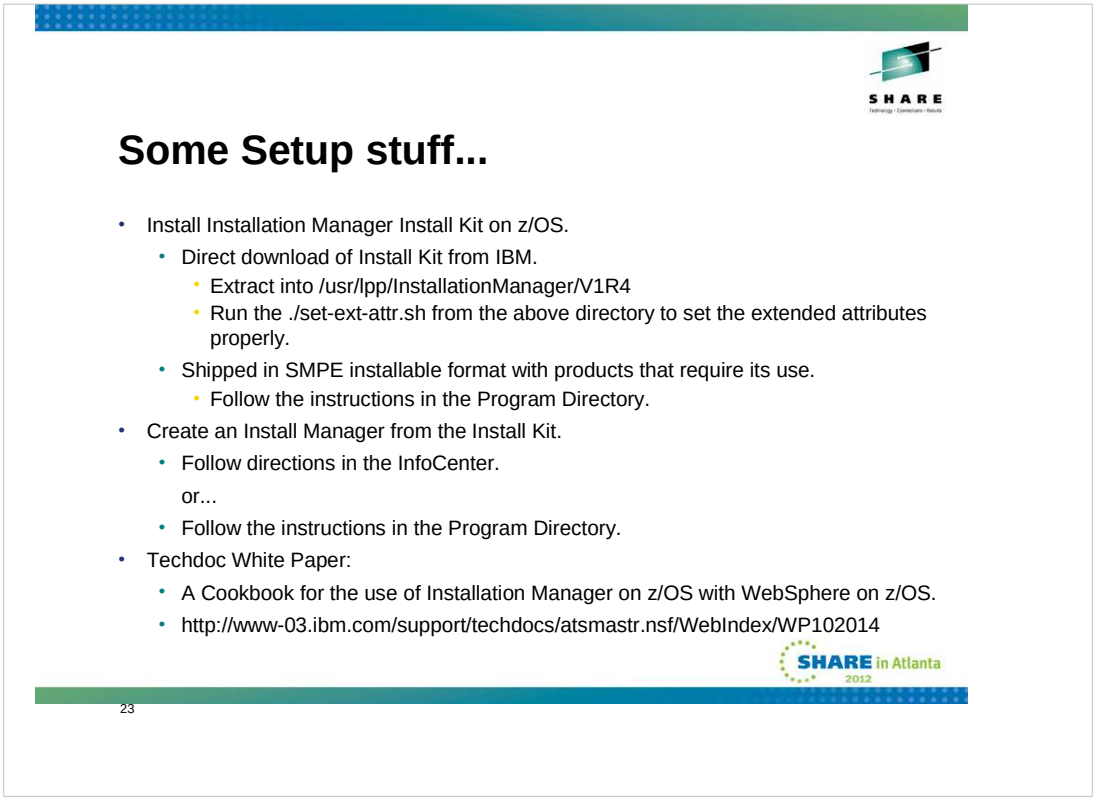


As we indicated earlier, the first thing you need to begin configuring WebSphere is to install the WCT on the workstation.

Before you can install the WCT, you need to have the Installation Manager installed on your workstation at an appropriate level. Once it is installed, unless you do something to prevent it, it will update itself to the most current level whenever it starts.

To install the Installation Manager, you have to download it (it is a zip file).

The url at the time this was written is:

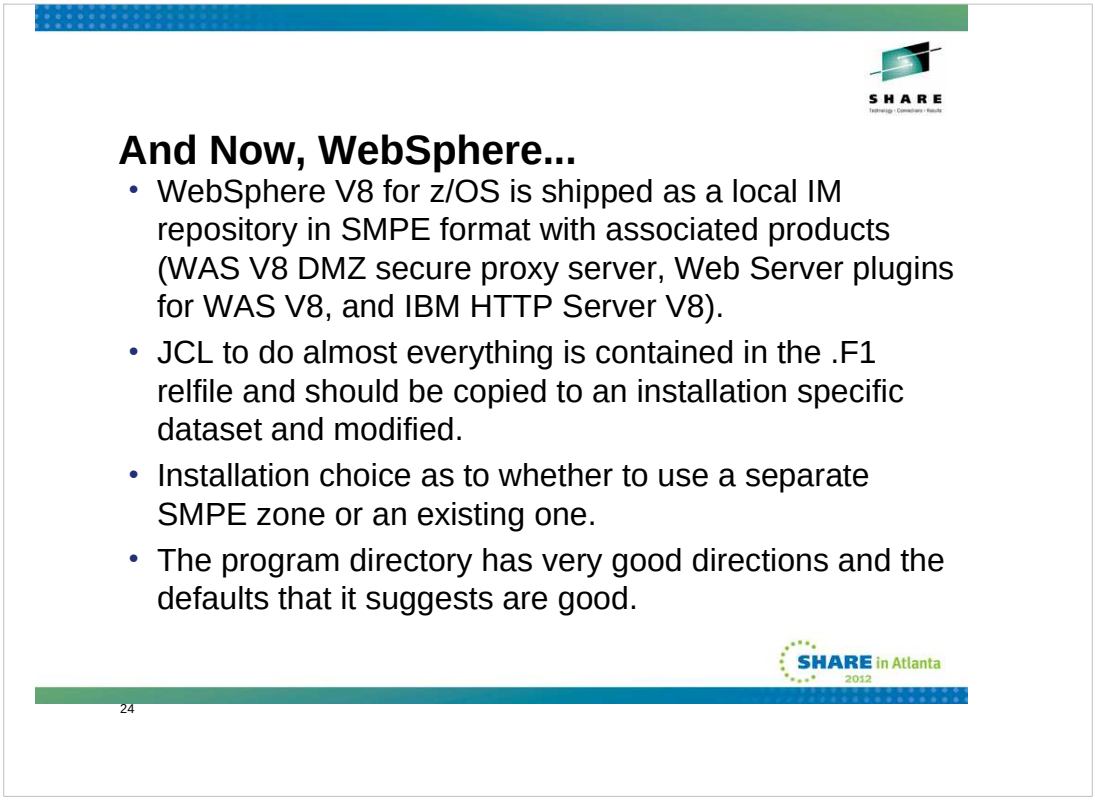


You will also nee to install the Installation Manager on z/OS. The first step in this process is to install the Installation Manager Install Kit.

The Install Kit may be either:

- Download the install kit (zip) directly from IBM.
- Extract the zip into (usually) /usr/lpp/InstallationManager/V1R4
- Run the provided ./sset-ext-attr.sh script to properly set the extended attributes.

Or

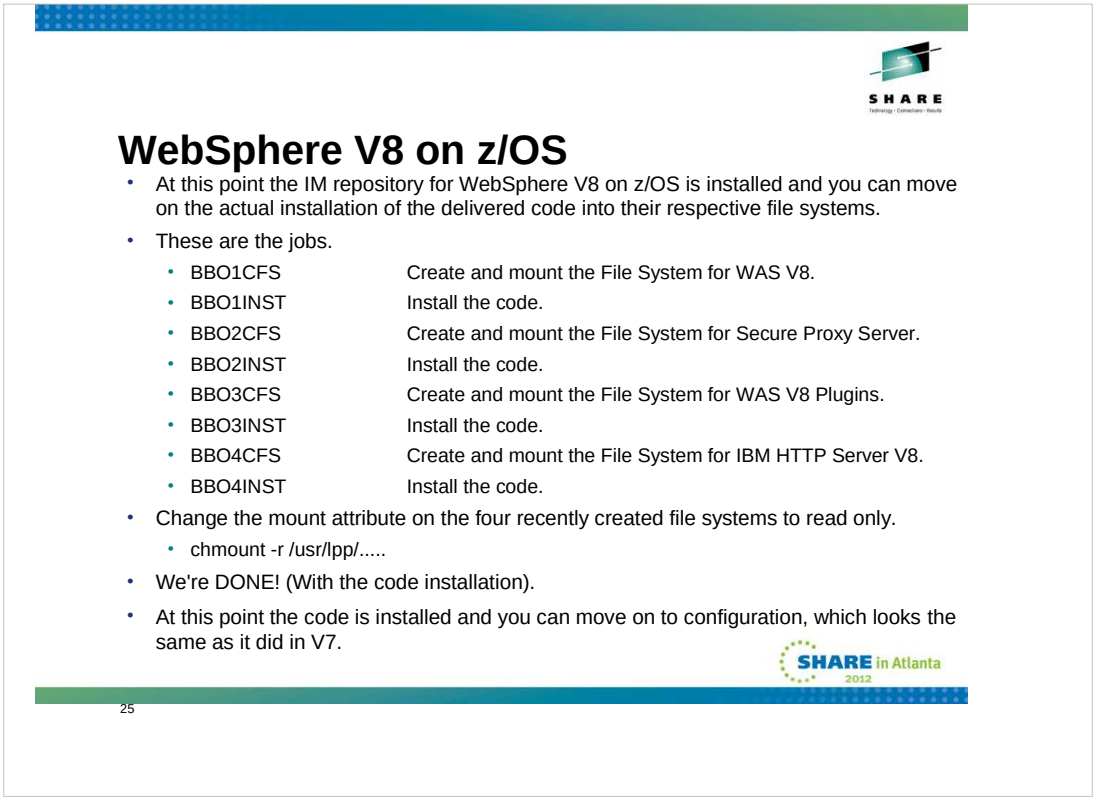


To actually install the WebSphere binaries, you start with a local IM repository that is installed with SMPE. It includes the WAS V8 for z/OS base code, the WAS V8 for z/OS DMZ Secure Proxy Server, the Web Server plugins for WAS V8 for z/OS, and the IBM HTTP Server V8 (apache based).

The JCL needed to use Install Manager to create all of the appropriate file systems and load them is included in the .F1 relfile and should be copied to your own dataset and modified to fit your installation standards. Basically the only thing you should have to modify is the jobcard and the names for filesystems, mountpoints, etc. The sizes for the artifacts are correct.

You have the choice of using an existing SMPE zone, or creating a new one for this repository. Up to you...

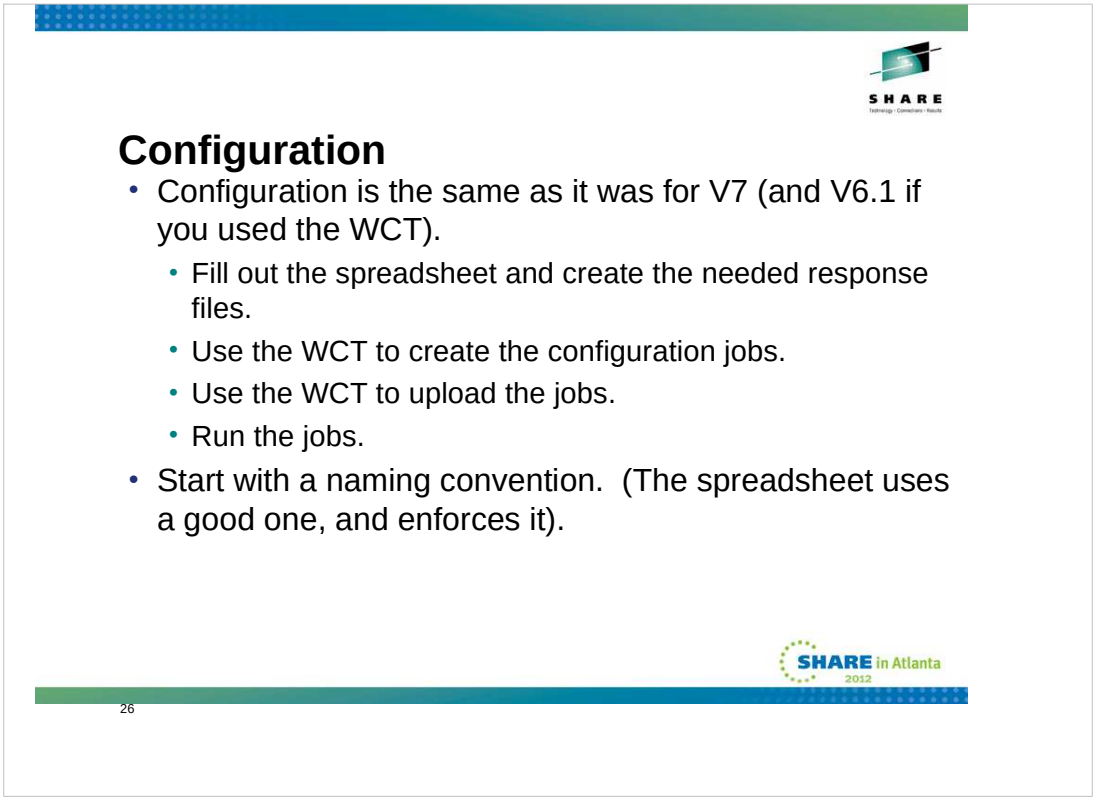
The program directory has very good directions and the defaults it suggests are reasonable.



Once the repository is installed (SMPE work completed), you can then use the Install Manager that you created previously to actually install the code into the appropriate file systems. There is a pair of jobs for each of the components in the repository.

Of course, while you are creating these, the file system must be mounted read/write, so once you are done, you should change the mount attribute to read only, as the example command shows.

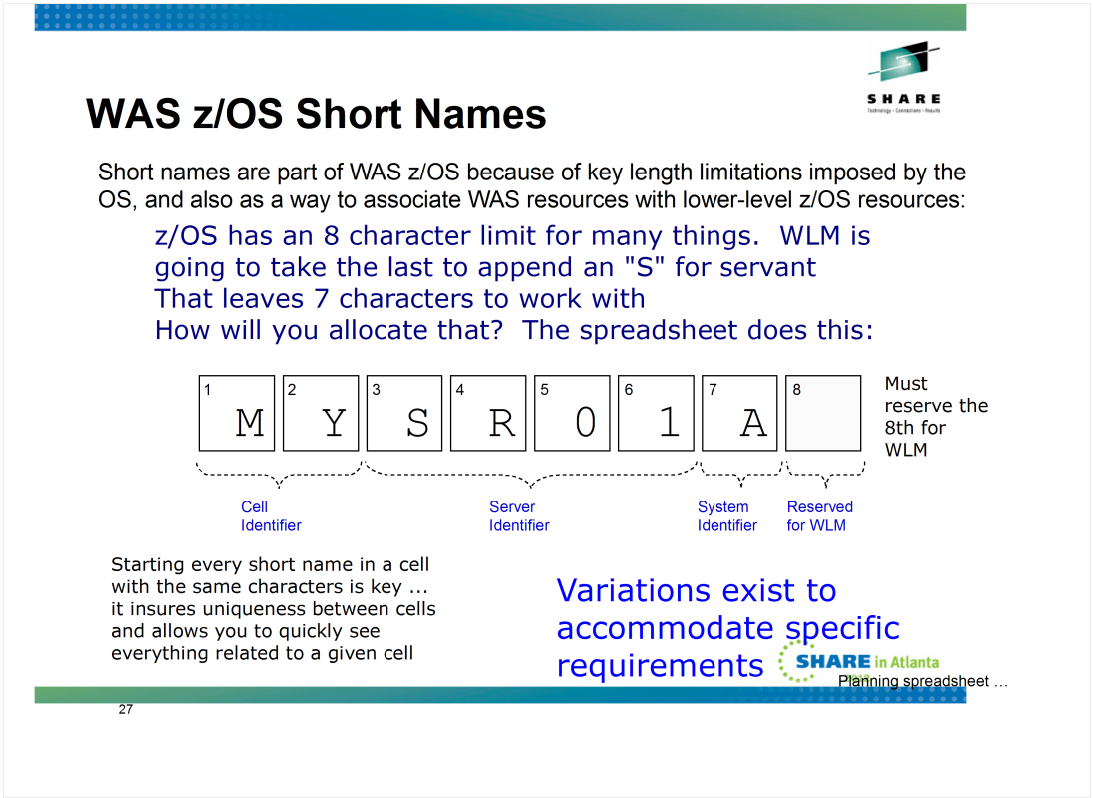
At that point you are ready to move on to configuration, which is basically unchanged from the previous release.



The sequence of events is:

1. Fill out the spreadsheet.
2. Use the spreadsheet to create the necessary response files.
3. Use the WCT (PMT) to create the configuration jobs using the response files as input.
4. Use the WCT to upload the jobs to z/OS.
5. Run the jobs.

It is imperative to start with a good naming convention which the spreadsheet uses and enforces. This is a STRONG recommendation, but unfortunately not a requirement.



Short names are used only by WAS z/OS. They came about because z/OS has key length limitations that need to be taken into account. In general the magic number is 8 ... many names and values associated with z/OS are limited to 8 characters. These include the JCL start procedure names, most SAF values, and the z/OS JOBNAME used for the started task.

We have determined the best place to start the planning process for short names is the controller JOBNAME value.

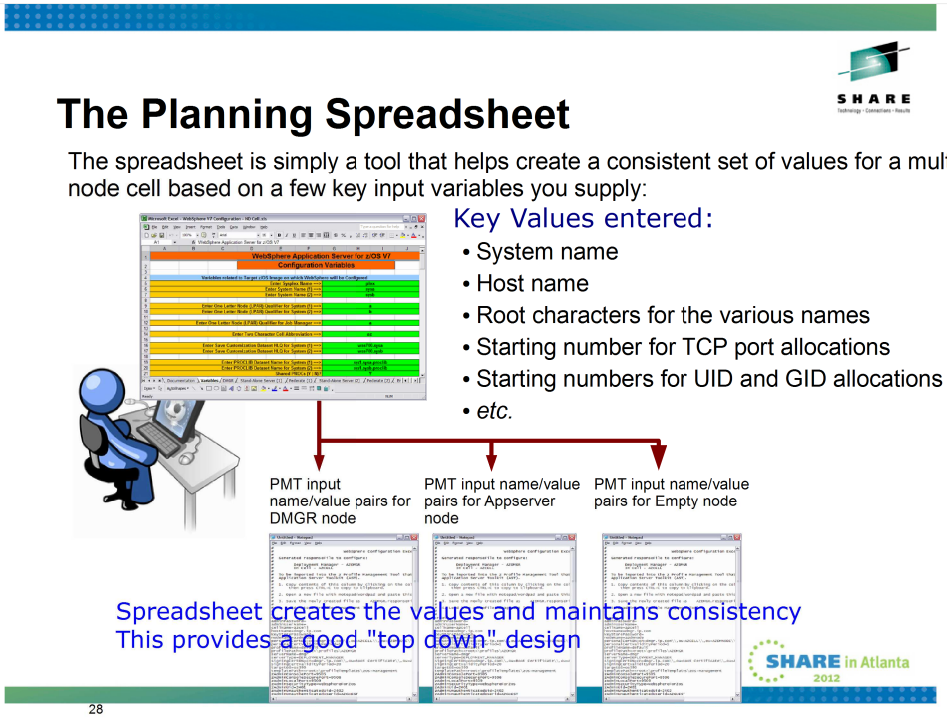
The key to this is understanding that WLM starts the servant region. And by doing so, it provides a JOBNAME for the servant. What it does is add an "S" to the controller JOBNAME. Therefore, the JOBNAME for the controller should be limited to 7 characters so WLM has the space to add the S for the servant.

Further, we have found that it's best to start all controller JOBNAMES with the same characters. What those characters are isn't so important, other than they adhere to z/OS standards such as starting alpha and staying within alpha, number and national characters.

The rest of the controller JOBNAME is allocated to some sort of identifier of the server in question. Again, the specific characters doesn't really matter, provided they're meaningful to you and there's some consistency between servers.

The last thing is an LPAR identifier. That occupies character 7.

You could plan this out by hand. Or you could use the planning spreadsheet. That's what we'll cover next.



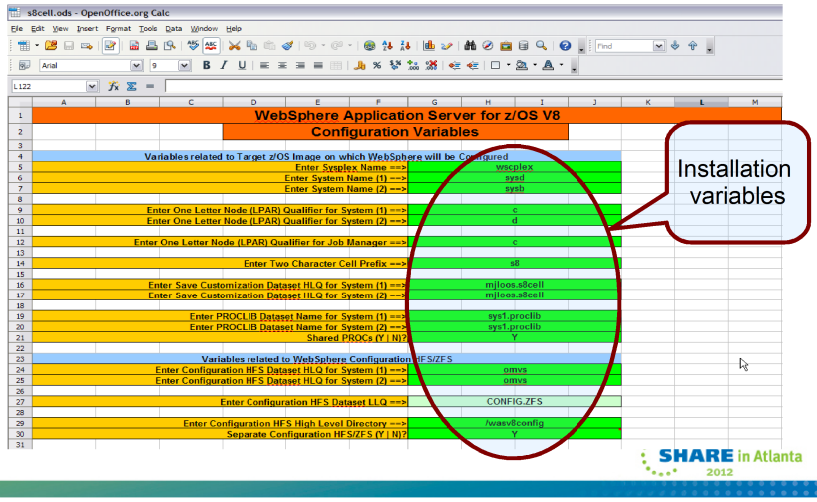
The "Planning Spreadsheet" is simply a spreadsheet (both Excel and OpenOffice) that takes a few key variables from you on one worksheet and *generates a consistent set of values for various PMT node options on other sheets.*

The produced output is a set of name/value pairs that serves as input to the PMT. It's simply a matter of copying the values from the spreadsheet and paste them into a text file. From there you import it into the PMT. The PMT then has *all the values it requires to build the node*.

Again, the key is that this produces a set of consistent values across nodes for a cell. There's no magic to it. It's simply programming within the spreadsheet that enforces the consistency.

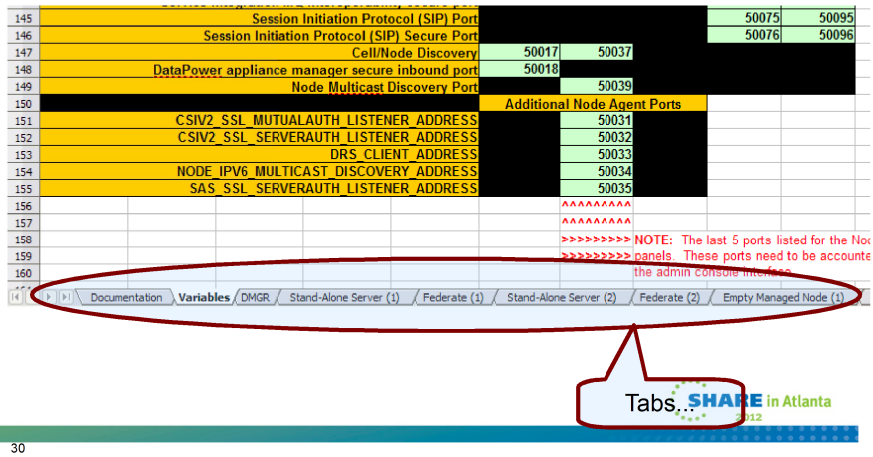
This is what we call a "top down" design. By that we mean a design that takes into account the end objective of the cell design, and keeps consistent those things that require it.

The Spreadsheet



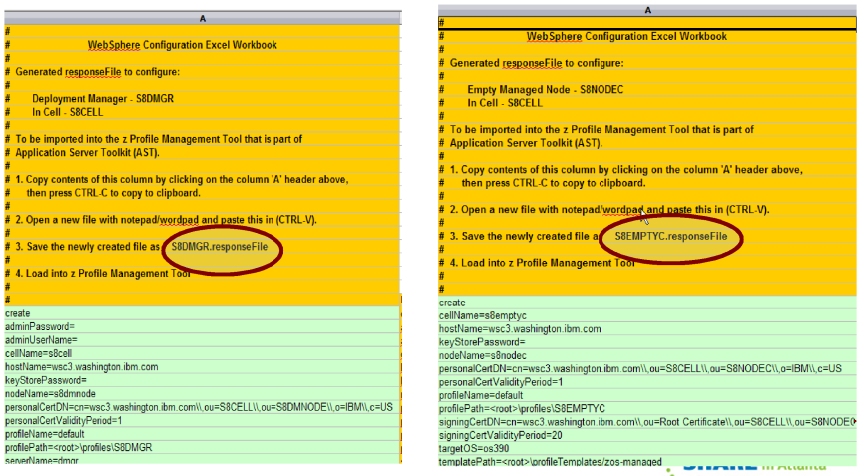
This is the variables tab of the spreadsheet that was used for the cell in this presentation.

The Spreadsheet



And these are the “tabs” which allow you to select the various response files.

The Spreadsheet – response files



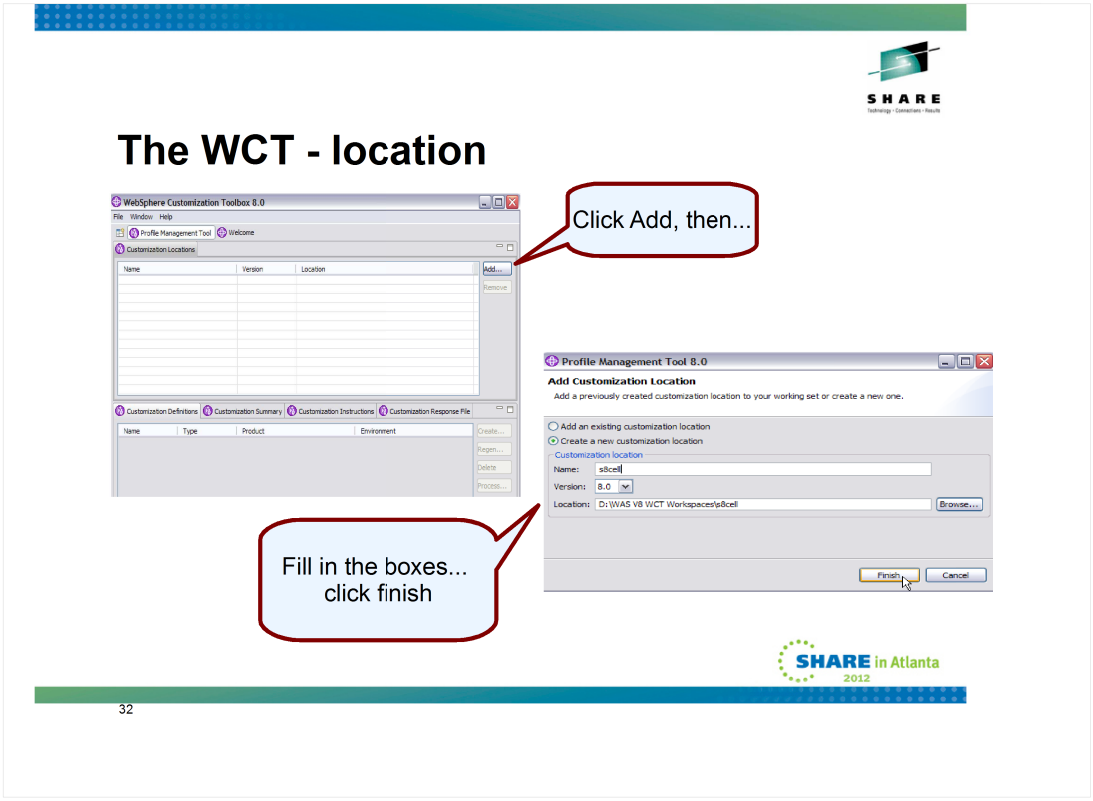
A

```
#
#                    WebSphere Configuration Excel Workbook
#
# Generated responseFile to configure:
#
#       Deployment Manager - S8DMGR
#       In Cell - S8CELL
#
# To be imported into the z Profile Management Tool that is part of
# Application Server Toolkit (AST).
#
# 1. Copy contents of this column by clicking on the column 'A' header above,
#      then press CTRL-C to copy to clipboard.
#
# 2. Open a new file with notepad/wordpad and paste this in (CTRL-V).
#
# 3. Save the newly created file as S8DMGR.responseFile
#
# 4. Load into z Profile Management Tool
#
#
create
adminPassword=
adminUserName=
cellName=s8cell
hostName=wsc3.washington.ibm.com
keyStorePassword=
nodeName=s8dmnode
personalCertDN=cn=wsc3.washington.ibm.com\\,ou=S8CELL\\,ou=S8DMNODE\\,o=IBM\\,c=US
personalCertValidityPeriod=1
profileName=default
profilePath=<root>\profiles\S8DMGR
serverName=dmgr
```

A

```
#
#                    WebSphere Configuration Excel Workbook
#
# Generated responseFile to configure:
#
#       Empty Managed Node - S8NODEC
#       In Cell - S8CELL
#
# To be imported into the z Profile Management Tool that is part of
# Application Server Toolkit (AST).
#
# 1. Copy contents of this column by clicking on the column 'A' header above,
#      then press CTRL-C to copy to clipboard.
#
# 2. Open a new file with notepad/wordpad and paste this in (CTRL-V).
#
# 3. Save the newly created file as S8EMPTYC.responseFile
#
# 4. Load into z Profile Management Tool
#
#
create
cellName=s8emptyc
hostName=wsc3.washington.ibm.com
keyStorePassword=
nodeName=s8nodec
personalCertDN=cn=wsc3.washington.ibm.com\\,ou=S8CELL\\,ou=S8NODEC\\,o=IBM\\,c=US
personalCertValidityPeriod=1
profileName=default
profilePath=<root>\profiles\S8EMPTYC
signingCertDN=cn=wsc3.washington.ibm.com\\,ou=Root Certificate\\,ou=S8CELL\\,ou=S8NODE
signingCertValidityPeriod=20
targetOS=os390
templatePath=<root>\profileTemplates\zos-managed
```


Looking at two of the tabs, we can see the deployment manager response file on the left, and the first empty managed node response file on the right.

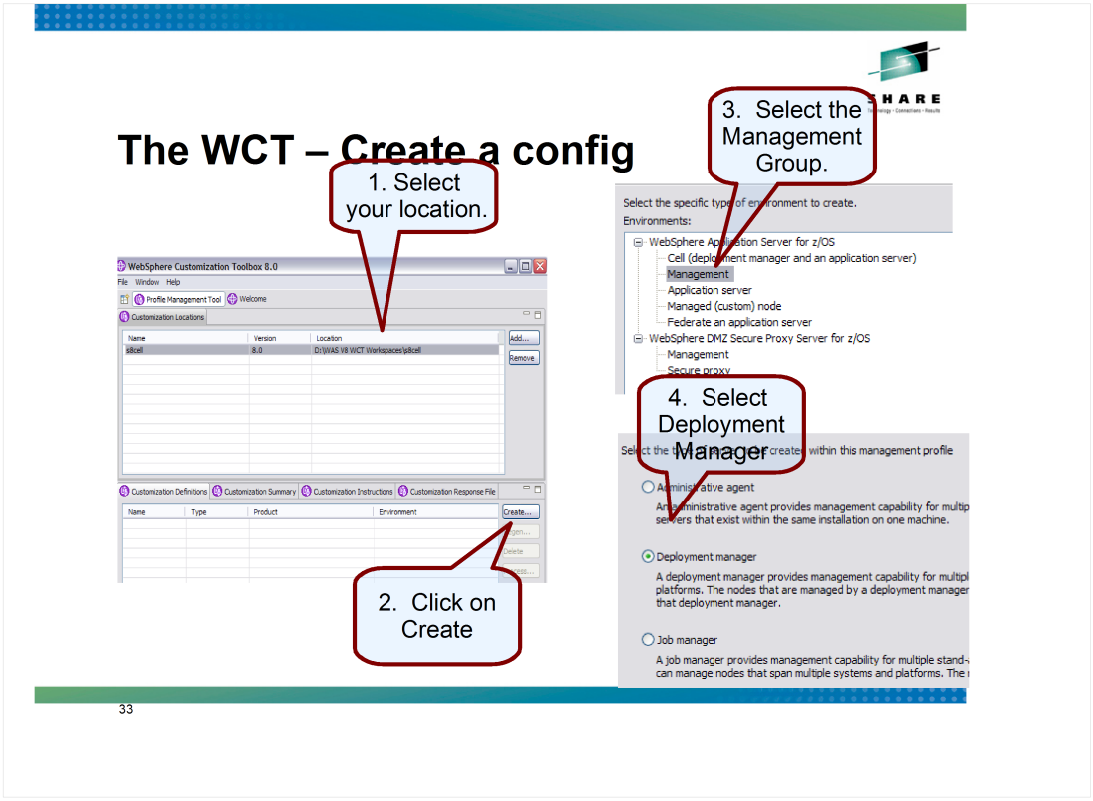


Once we've started the WCT, the first thing we'll need to do is add a location. A location is just what it sounds like. A named area on workstation disk that is used to store all of the elements of a configuration.

To create a new one, you click on Add, then fill in the blanks on the next panel.

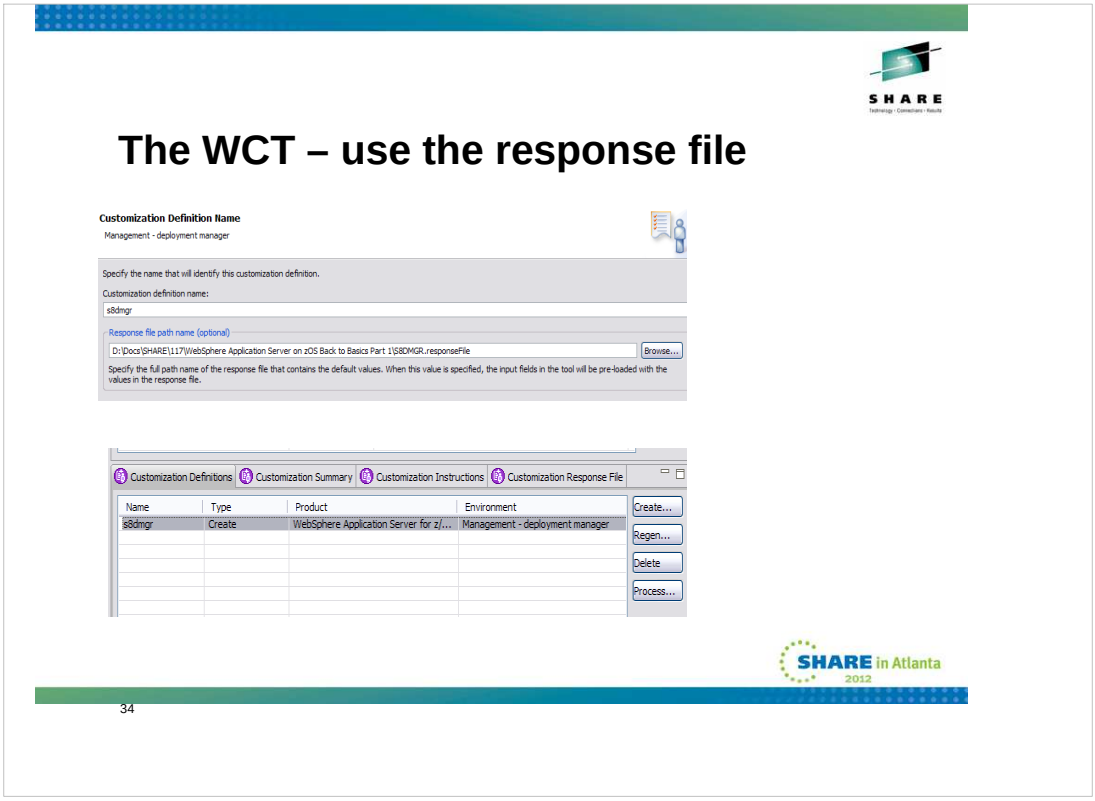
Name for the location in this case is the cell name, you select the version in the drop down box, and then indicate where on disk you want the location to reside.

If you are adding an existing location, you would simply browse to the correct



Once you have a location, you can start to create configurations.

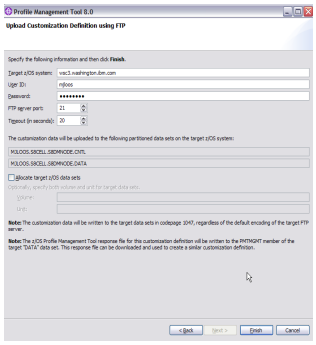
In this case we are going to create a deployment manager, so you first make sure you have the correct location highlighted, then click on Create, then select the management suite, then select deployment manager on the next panel.

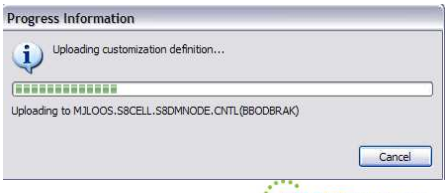


On the next panel you'll be given the opportunity to “name” your configuration. This isn't anything that will “carry through” but is how this configuration is named within the WCT. It is on this panel that you are allowed to point at a pre-existing response file. In our case, we'll point at the one we save for the deployment manager out of the spreadsheet.

Now if you accept what the spreadsheet has done on your behalf, you'll simply click next a lot until you get to the end. If you aren't using the spreadsheet, or feel

The WCT – upload jobs

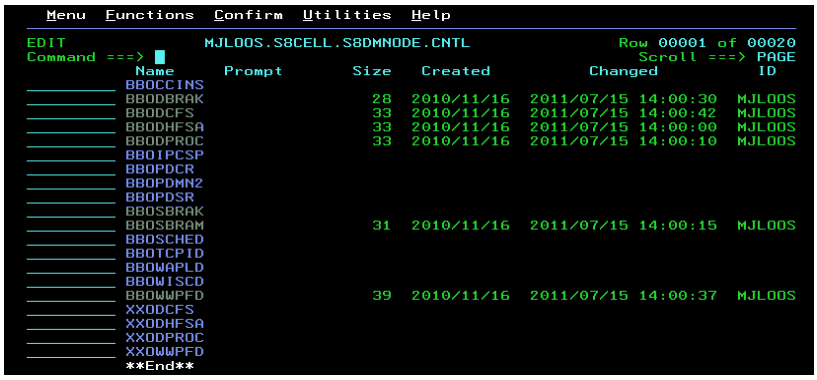




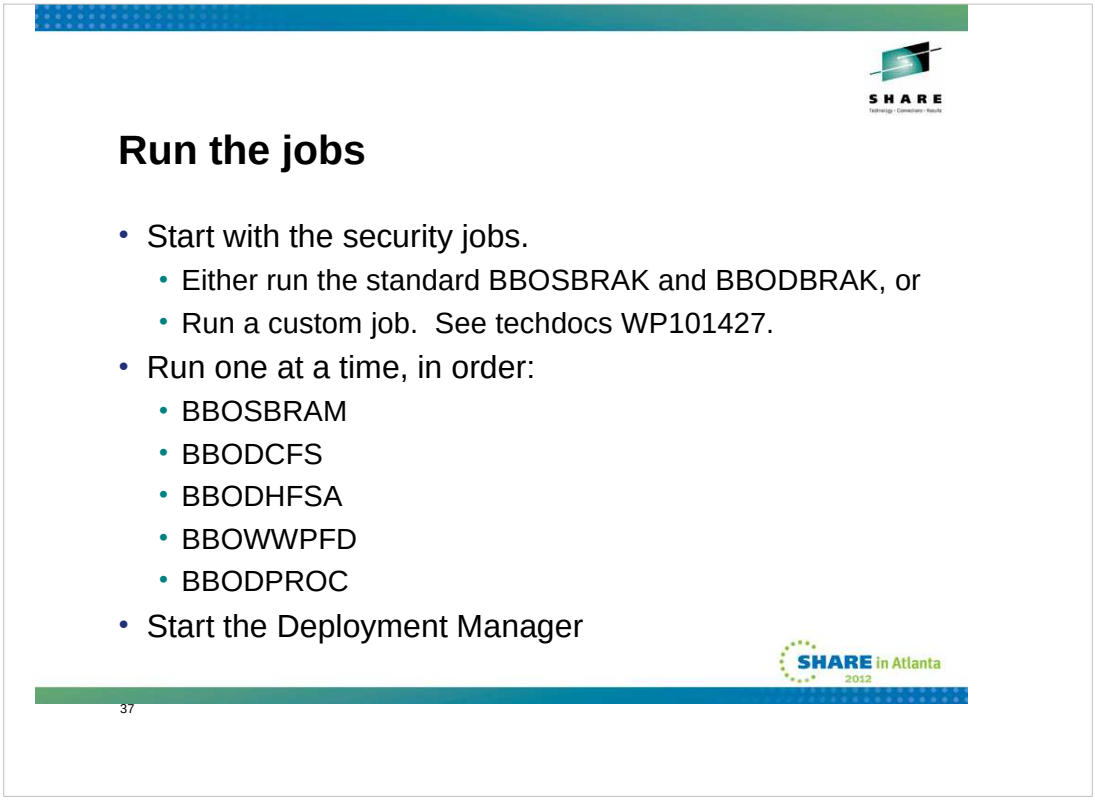
SHARE in Atlanta 2012

Next you will select the option to Process the configuration and essentially be guided through uploading the jobs that are the result of the the create process to the z/OS system where the configuration will reside.

Run the jobs



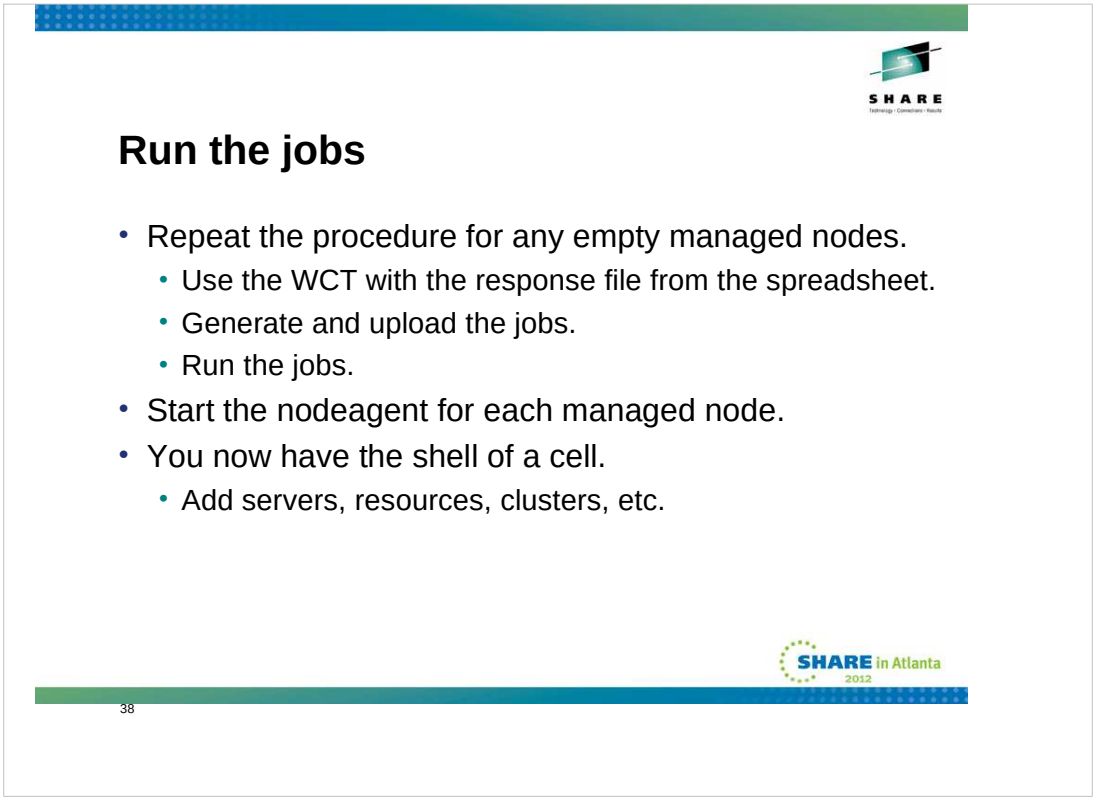
This is a member list view of the CNTL dataset which was uploaded for the deployment manager build process, with the members containing the jobs needed to build the configuration highlighted.



To actually build the configuration, you will run the jobs, one at a time, in the order the instructions (from either the WCT or the BBOxxINS member of the CNTL dataset) specify.

First you'll be running either the BBOSBRAK and BBODBRAK jobs to define all of the necessary SAF profiles for the configuration.

Alternatively, you have the option of running a custom job. There is a techdoc that describes in detail an alternative that builds a set of generic profiles that will cover the entire cell



Once the deployment manager is up and running (you must leave it running to complete the federation jobs in the empty managed node configuration jobs), you can repeat the process for each of the managed nodes that you have decided you need.

After each set of jobs for the empty managed nodes are completed, if you didn't allow it to happen automatically, you'll need to start the nodeagent for the node.

At this point, you have a cell with a deployment manager node and server

Creating an additional Managed Node

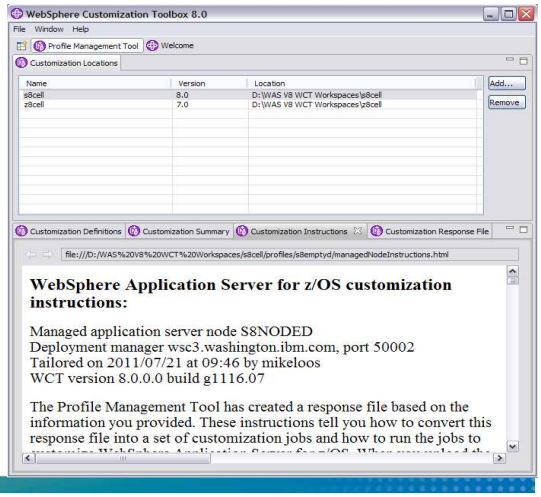
- For this section of the PMT (Empty Managed Node):
  - Load the correct “response file”.
  - Step through the panels.
    - Make any changes that seem necessary.
  - Create the jobs.
  - Upload the jobs.

Using “Simplified RACF definitions…

- Two Jobs in the .CNTL datasets run a REXX Exec in the .DATA datasets.
  - BBOSBRAK ➔ BBOSBRAC
  - BBOMBRAK ➔ BBOMBRAC
- The two REXX execs may be modified or replaced.
  - We used a set of modifications when this cell was originally created, illustrated here.
    - BBOMBRAK ➔ DORAC700
- Some documentation on the modifications…
  - Using Generic RACF Profiles with WebSphere on z/OS V7
  - http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101427

Creating an additional Managed Node

- Read the instructions.



Generated Instructions.

Creating an additional Managed Node

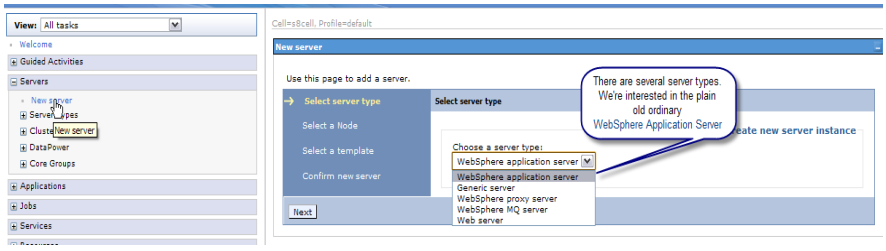
- Running the generated JOBs.
  - BBOMCFS
  - BBOMHFSA
  - BBOWWPFM
  - BBOMPROC
  - BBOWMNAN
- Start the nodeagent.

Note the lack of any security related jobs…

We used the generic RACF definitions described in the white paper and everything was defined when the cell was first created so nothing needs to be done at this time.

Creating a new server

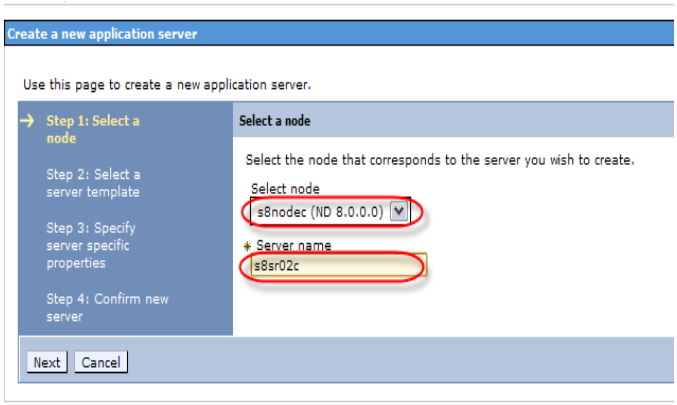
- Create a new server…



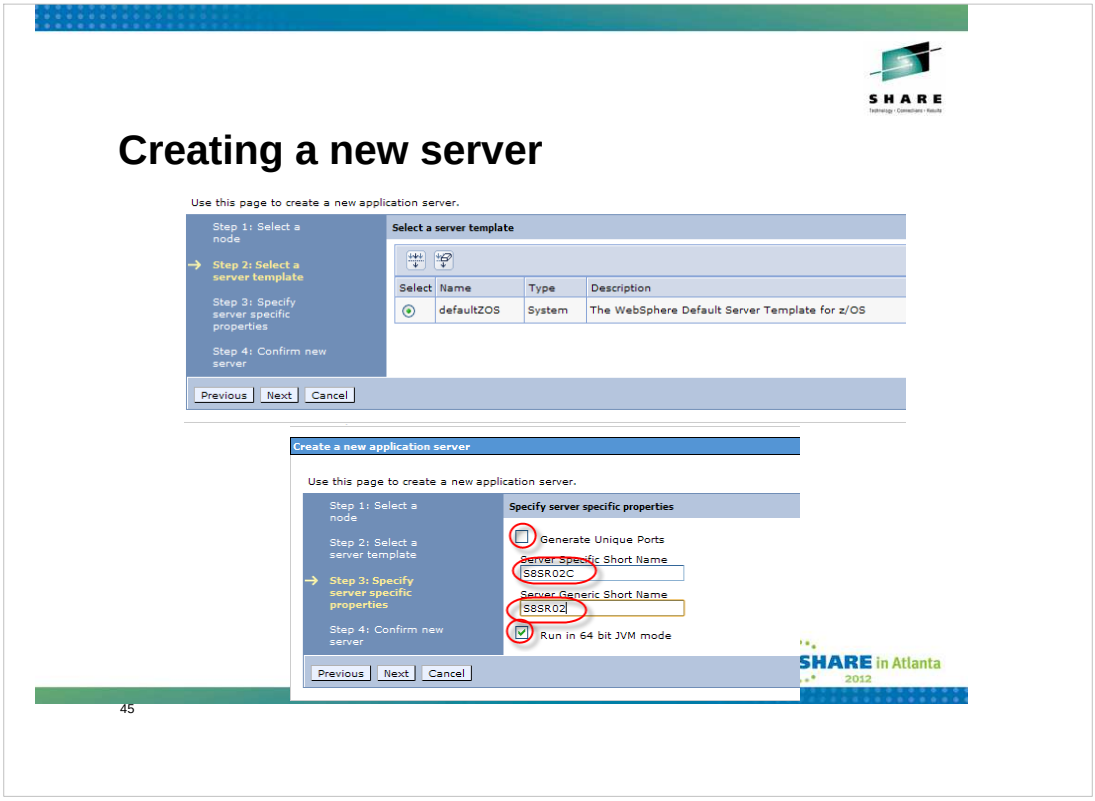
To create a new server.

Click on Servers >> New Server, select the server type (the default of WebSphere Application Server is what we want for this exercise), and click on Next.

Creating a new server



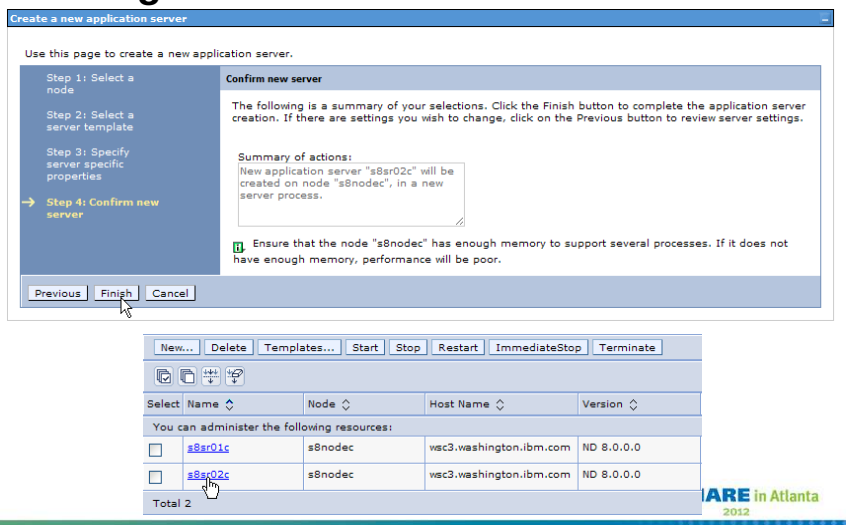
Select the proper node from the dropdown box (we only have one at this point so it is an easy decision) and specify the new server name. Then click on Next.



Select a template (the only one available at this point is the default template). Click Next.

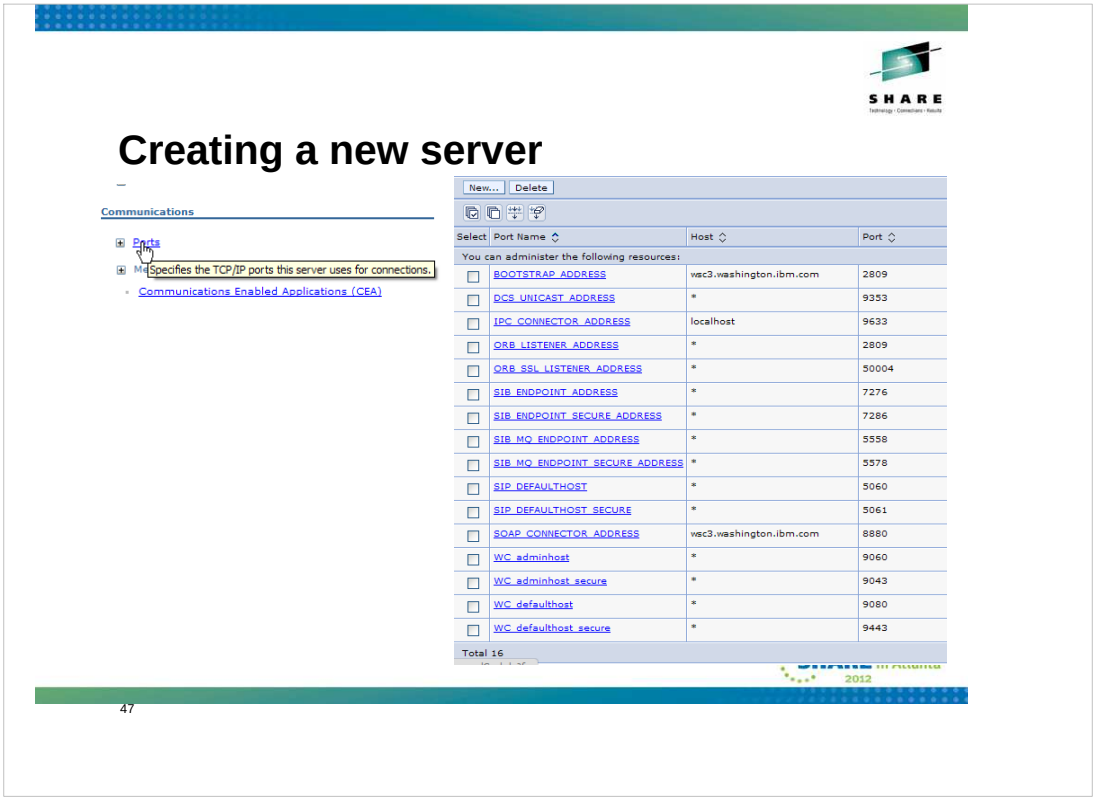
Uncheck the “Generate Unique Ports” box, specify the Server Short names (Specific and Generic), and if you don’t want to run in 64 bit mode (which is the V7 default) uncheck the box. We’ll leave it checked. Click Next.

Creating a new server



You will next be presented with a confirmation panel. Click on Finish.

Then click on your new server.



On the next screen, locate the section for Ports and click on Ports.

You will be presented with a screen listing all of the ports for the server. You now need to set each port to the proper number based on your numbering scheme.

You may also have to add some additional host alias entries to the default virtual host for the http ports.

You may then click on Save, and synchronize and your new server is ready to start.

Creating a new server

- As an alternative to setting the ports manually…
- A script!

updNewServerv8.py

As an alternative to setting the ports manually, you may wish to create and run a script to set them all manually after the server had been saved and the configuration synchronized. The script shown uses the WSC Naming convention, but could easily be adapted to other naming and numbering conventions…

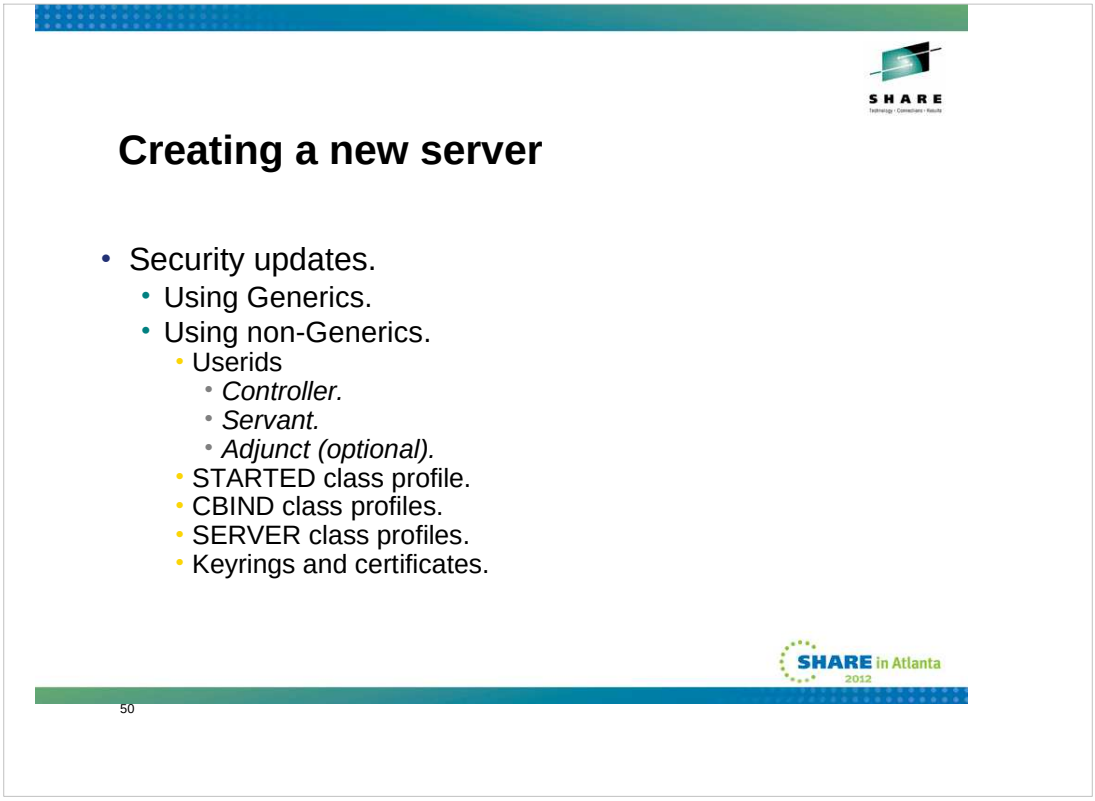
As an added benefit, it adds the necessary virtual host host alias entries as well.

Creating a new server

- And if you really want to avoid the adminconsole…

createNewServerv8.py

As an alternative to creating the server manually, you can run a script to do the entire process. The script shown requires that you tell it as arguments the servername, the nodename, and the originating (or low) port for the server. If you were using the WSC naming convention, the script would require no modification as it would properly set the specific and generic server short names and the port specified would be the soap port for the server. Again, if you are using a different naming convention the script should be fairly easy to modify to fit your needs.



Security profiles may need to be updated. If you have defined a generic set of RACF definitions which will cover these new servers, you may not need to change anything at this point.

In a cell where the RACF definitions are generic, it is common to use one USERID for all controllers and one for all servants and adjuncts. In this case the keyring and certificate that is defined for those USERIDs will suffice.

However, if you used the default RACF definitions, which are specific (not generic), as provided by the BBOWBRAK Rexx exec in the generated .DATA file, you will need to add the following:

New USERID for each of the address spaces (controller, servant, and optionally adjunct).

STARTED profile for the new server controller, servant, and optionally the adjunct regions

CBIND and SERVER classes profiles to include the new Cluster Transition Name and the new server

Keyrings for the controller and servant (and possibly adjunct) regions, and new certificate for the controller (and possibly adjunct), and connect the CERTAUTH certificate to the keyrings.

Creating a cluster

- What is a cluster?

Artifact	*Name*	*Comment*
Cell	**s8cell**	**Existing**
Node on SYSC	**s8nodec**	**Existing**
Node on SYSD	**s8noded**	**Existing, Federated, Empty**
Cluster	**s8sr02**	**To be created**
Server on SYSC	**s8sr02c**	**Existing**
Server on SYSD	**s8sr02d**	**To be created**

SHARE in Atlanta 2012

51

A cluster, in its most simple form is a server. And every server is a cluster. But a cluster composed of a single server isn't of much interest. Usually when discussing a cluster, we are talking about a cluster containing two or more servers, generally in more than one node.

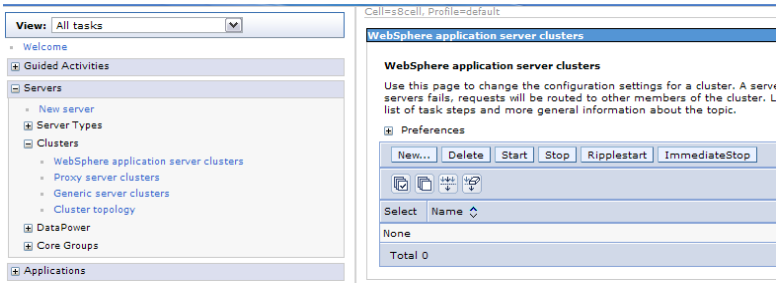
Creation of a cluster can start with zero, or one servers or templates. A cluster can never be created from more than one server.

Our starting point for this exercise is the table shown.

We have created so far in this presentation a multinode, mutisystem cell. The cell is the s8cell, and we have a node s8nodec on SYSC and a node s8noded on SYSD (which we have just created). The s8nodec node has a server s8sr02c (also recently created).

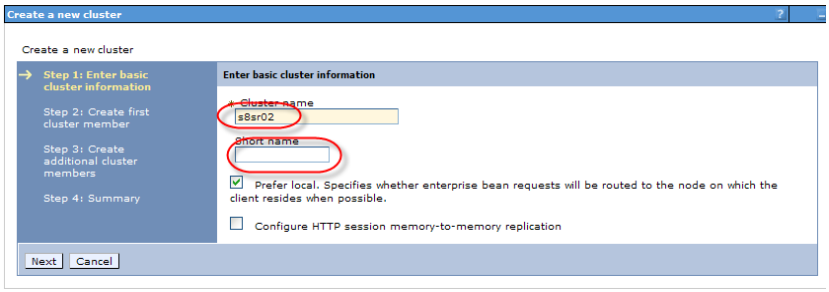
We will create a cluster s81sr02 that will consist of s8sr02c which will be converted into a cluster member, and we will add one additional member s8sr02d in node s8noded.

Creating a cluster



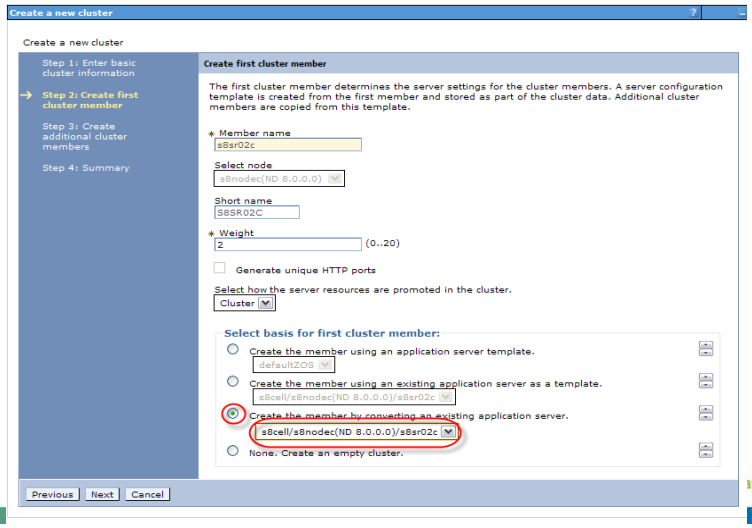
As always, we start by logging on to the adminconsole. Then we navigate to Servers >> Clusters >> WebSphere application server clusters and then click New.

Creating a cluster



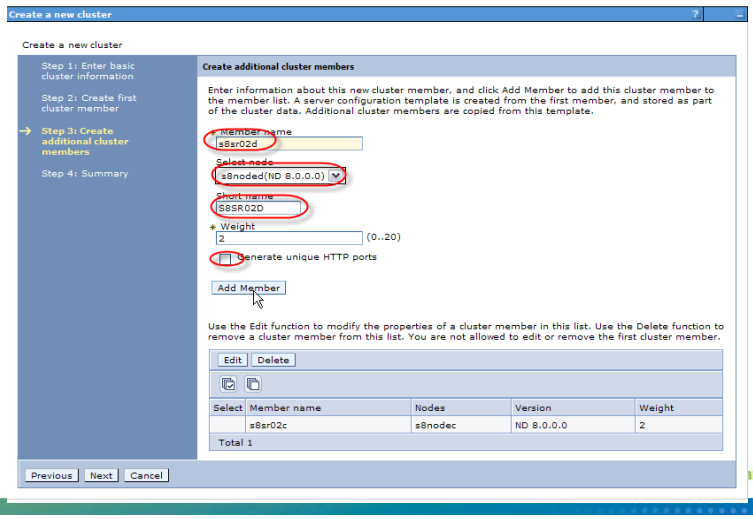
You'll see a screen similar to the one shown where you can fill in the values for Cluster name. We leave the cluster Short name blank. This will make more sense after the next foil. Then click Next.

Creating a cluster



Now we're ready to create the first cluster member. We're going to go the route of creating the member by converting an existing application server. As soon as we select that option, many of the other fields "grey out". We select s8sr02c in node s8nodec from the drop down box after checking the radio button. The cluster we are creating will inherit it's short name (left blank on the previous screen) from the server we are converting to a cluster member. Then click Next.

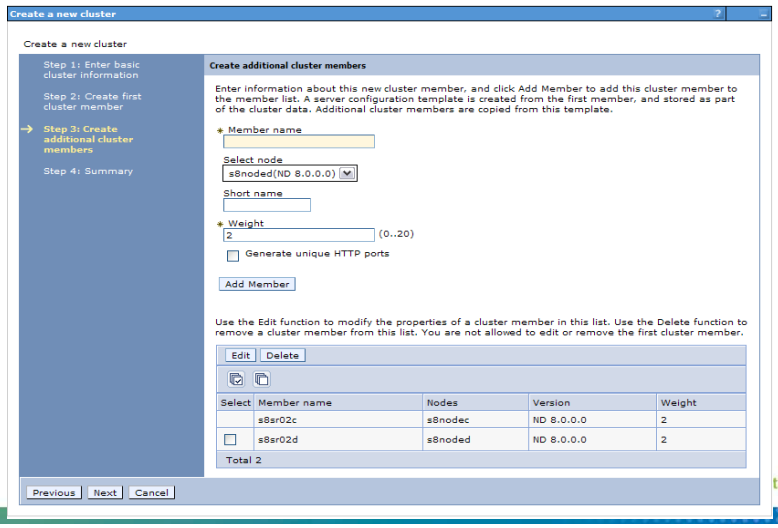
Creating a cluster



55

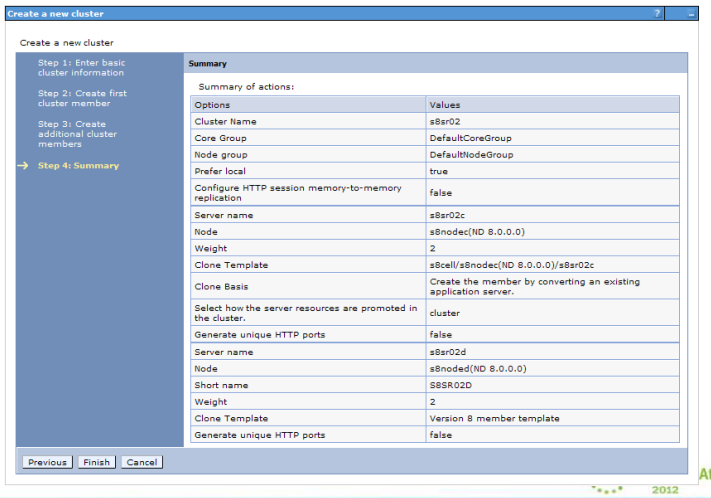
This is the next screen you'll see. Fill in the member name of the new cluster member (the one that doesn't yet exist), s8sr02d, select the appropriate node from the drop down box, in this case s8noded, and fill in the short name, S8SR02D in our example. Uncheck the Generate unique http ports (in most cases). Then click on Add Member

Creating a cluster



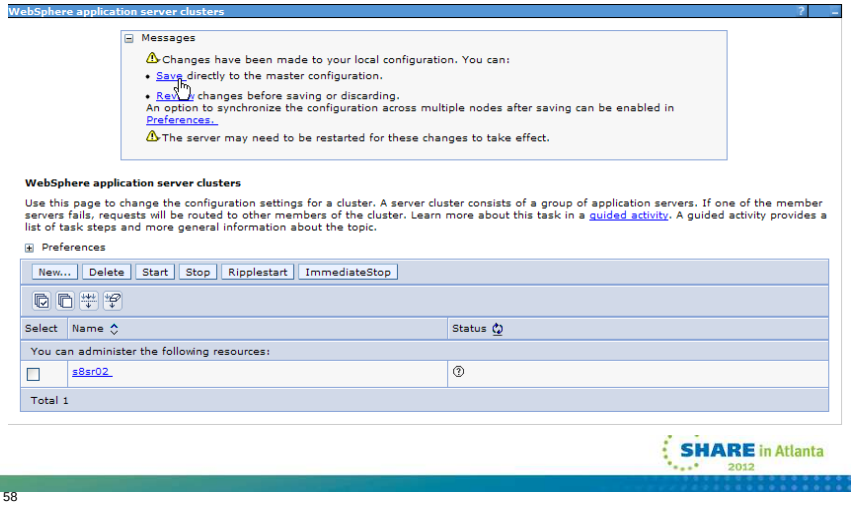
You'll be presented with a panel similar to this one. You'll notice that there is an additional cluster member in the table at the bottom of the panel. You now have an opportunity to add additional cluster members, simply by filling in the blanks as we did on the previous screen and clicking on add member. Two members is enough for our example, so we'll simply click on Next.

Creating a cluster



You'll see a summary page describing the to be created cluster. All you need to do is to click on Finish.

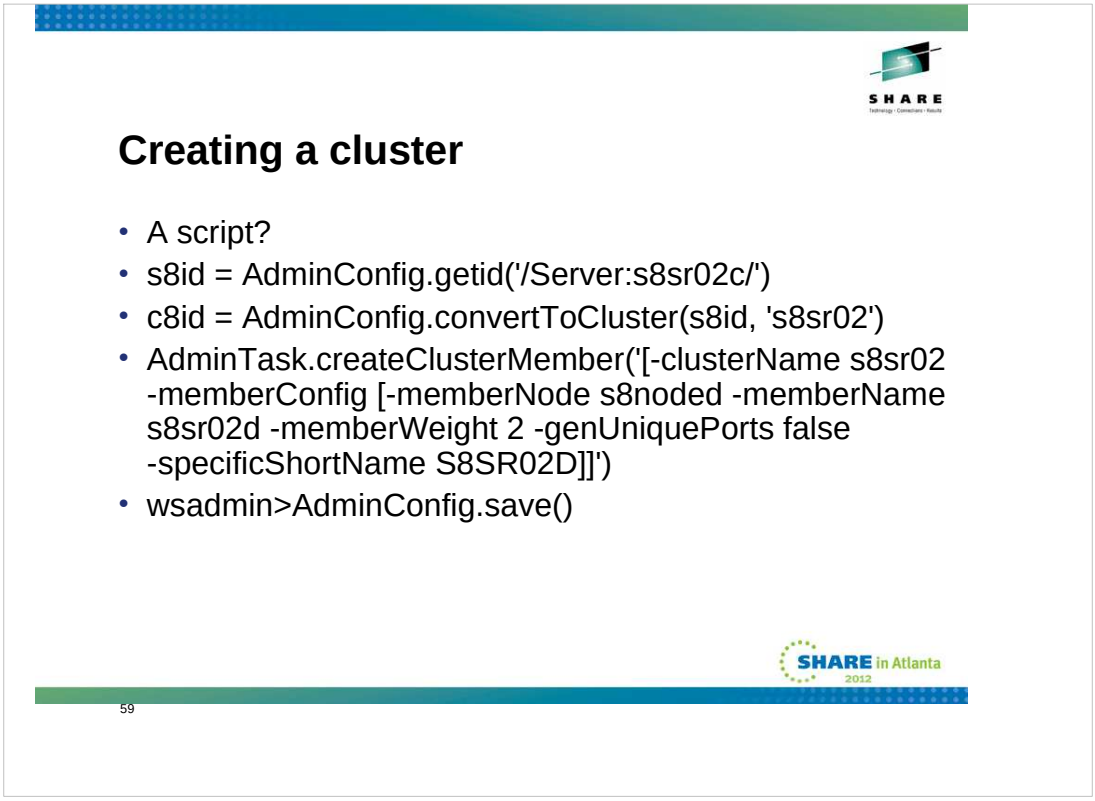
Creating a cluster



Simply Save and synchronize and you've created the cluster.

You may have to update the Host Aliases for the Virtual Host. If you have been using an asterisk(*) for the host name on the original server, this will not be necessary.

All that remains is to start the cluster members (servers) and verify that they start and run properly.



If you'd rather create your cluster using a script, here are the commands (bare bones) that you'd need.

The first gets the id of the server you choose to convert to a cluster.

The second converts the server with long name s8sr02c to a cluster with cluster name of s8sr02 with a single member named s8sr02c

You're essentially done at this point, but since the main reason for a cluster is to have two or more members…

The third command will create a new cluster member in cluster s8sr02 with member name (long name) of s8sr02d on node s8noded with a shortname of S8SR02D. This command also leaves the ports alone. You'll have to do something about that later.

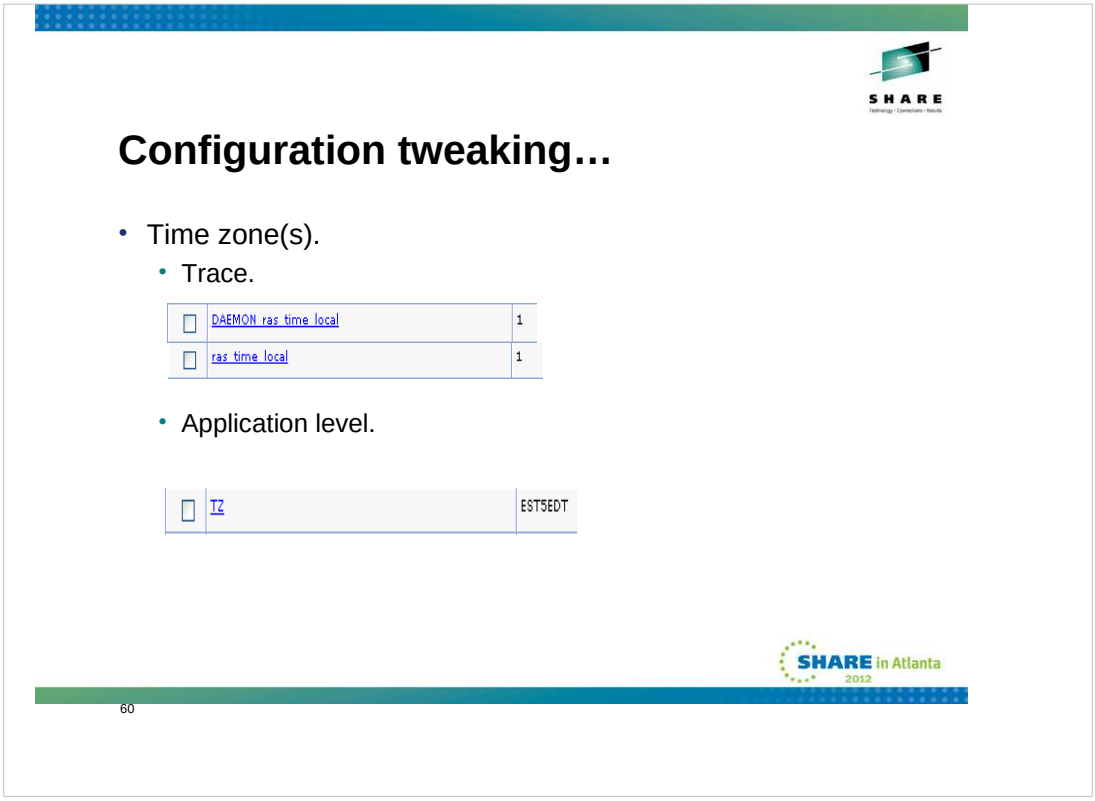
The last command saves the configuration.

Simple? Yes!

Now you should probably update the port assignments of the new cluster member, possibly using the previously mentioned script updNewServerv8.py.

You may have to update the Host Aliases for the Virtual Host. If you have been using an asterisk(*) for the host name on the original server, this will not be necessary.

All that remains is to start the cluster members (servers) and verify that they start and run properly.



The default for timestamps is for all of them to appear in GMT. Many installations would prefer to have the timestamps appear in local time. There are three variables which you may alter to change the behavior to match your desires. They are:

DAEMON_ras_time_local
ras_time_local

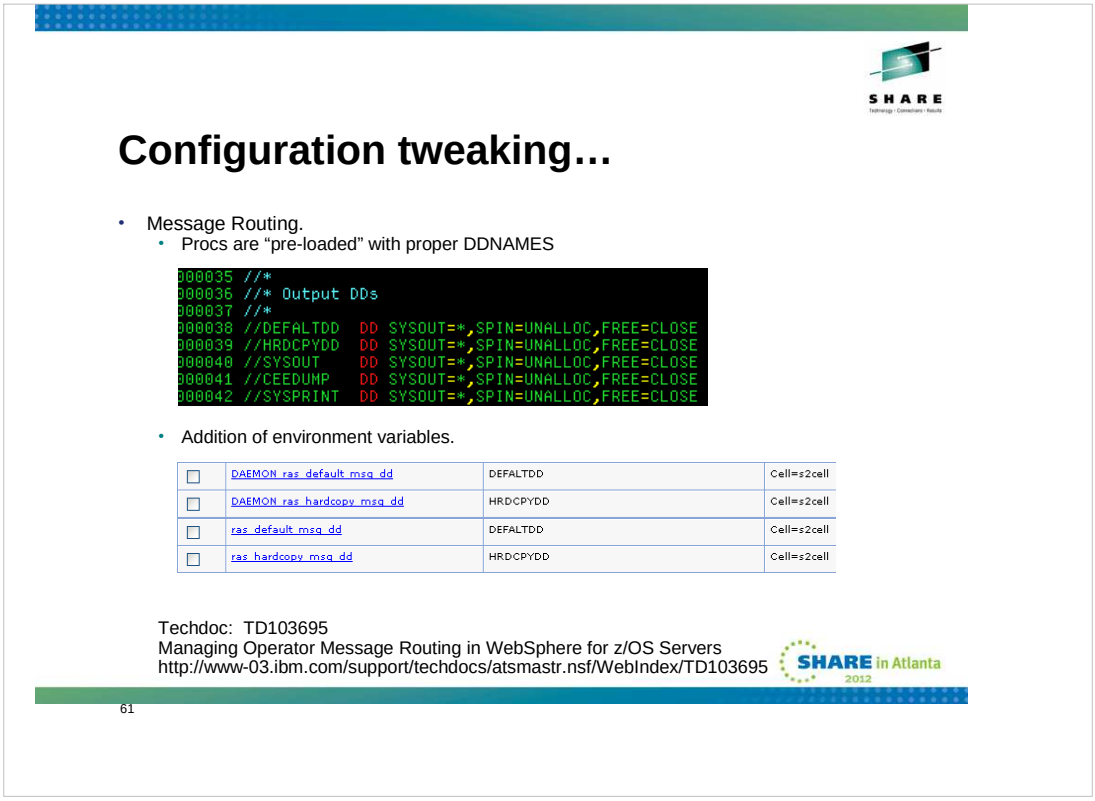
And **TZ**

DAEMON_ras_time_local and **ras_time_local** are set to **0** (GMT) by default. Actually the variables do not even exist. If you wish to change them, you have to add them and change their value to **1** (local).

The **TZ** variable also does not exist, and in its absence, all application time will be in GMT. This variable can be set to one of the standard timezone values, such as **CST6CDT** or **EST5EDT** or whatever is appropriate for your installation.

All of these variables are set by logging on to the adminconsole and navigating to the Environment >> WebSphere variables >> and adding them. I would suggest setting them at the cell scope and if you need to vary from that for a particular server you can add another instance at that server's level.

Then save the config and synchronize and the next time a component is started it will take effect.



As delivered, many messages are routed to the system log via a WTO. They also appear in the JESMSGLG and JESYSMSG DDs for the started tasks. The volume of messages on the syslog and coming across the console is annoying (to say the least). Fortunately it is a simple matter to change this behavior.

The routing of these messages is controlled by the following variables:

DAEMON_ras_default_msg_dd
DAEMON_ras_hardcopy_msg_dd
ras_default_msg_dd
ras_hardcopy_msg_dd

These variables may be set to the DDNAME that you wish to have them routed to and they will no longer be issued as a WTO. Again, the suggestion is to set them at the cell level and then if you need to change them on a finer grained level you can do so by adding a copy of the variable at a “lower” scope.

The variables are set by logging on to the adminconsole and navigating to the Environment >> WebSphere variables >> and adding them.

Then save, synchronize, and as each component is restarted, the change will take effect.

The techdoc describes additional controls that may be added that apply to JES2 systems.

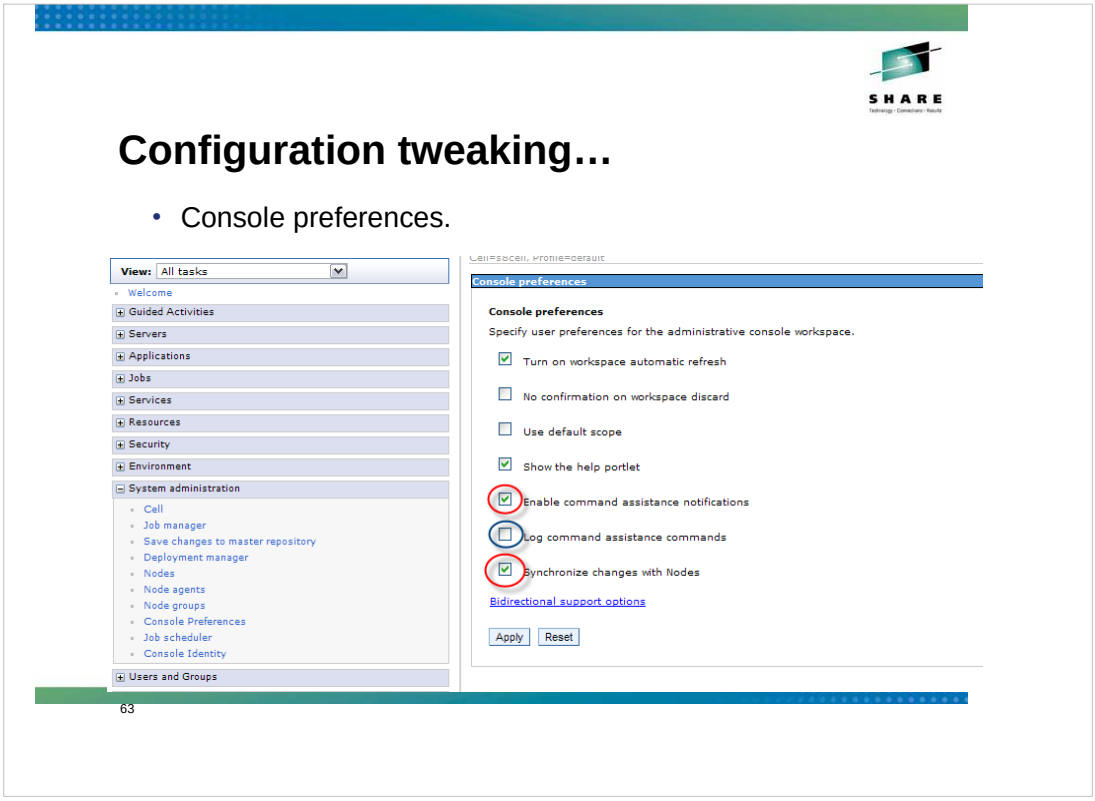
Configuration tweaking…

- An alternative to all of the editing…
- A script!

initsetvarsv8.py

Script is available at the following URL:
http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100963

If you are adverse to doing all of that manual editing, or if you create a lot of cells and wish to make them all look alike, a script might be a good alternative to doing all of the manual edits. The script I am showing, **initsetvarsv8.py** is a jython script which does all of the changes we made in the adminconsole (with the exception of the console preferences, since they are “per userid”).



One of the first things you may want to do when you first log on to the adminconsole is set your default console preferences. These tend to be individual preferences and I'm showing how I like them set. Your selections may of course vary…

You set them by logging on to the adminconsole, and navigating to: Systems administration >> Console Preferences

I suggest setting the Synchronize changes with Nodes on, and also like to have the command assistance notifications enabled.

You may also consider turning on the Log command assistance commands option.

Use of RMI Connector with wsadmin.sh

- wsadmin.sh uses either a SOAP connection or an RMI connection.
- The SOAP connection necessitates the passing of a clear text userid and password.
- The RMI connection can either prompt or use the userid and password of the logged on user.
- The RMI connection has been “observed” to be a little quicker than the soap connection (anecdotal).

Use of RMI Connector with wsadmin.sh

- To switch from the SOAP (default) connection to RMI…
- From somewhere that supports editing ASCII encoded files.
  - SSH, Telnet, etc.

```
cd /wasv8config/s8cell/s8mnode/DeploymentManager/profiles/default/properties
```

Use of RMI Connector with wsadmin.sh

Use an ASCII editor to change:

sas.client.props

From:

```
com.ibm.CORBA.securityServerHost=
com.ibm.CORBA.securityServerPort=
com.ibm.CORBA.loginSource=prompt
```

To:

```
com.ibm.CORBA.securityServerHost=wsc3.washington.ibm.com
com.ibm.CORBA.securityServerPort=50003
com.ibm.CORBA.loginSource=none
```

Use of RMI Connector with wsadmin.sh

Optionally, (these could be specified on the wsadmin.sh command line as parameters) use an ASCII editor to change:

wsadmin.properties

From:

```
com.ibm.ws.scripting.connectionType=SOAP
#com.ibm.ws.scripting.connectionType=RMI
com.ibm.ws.scripting.port=15610       /* from SOAP port */
com.ibm.ws.scripting.defaultLang=jacl
```

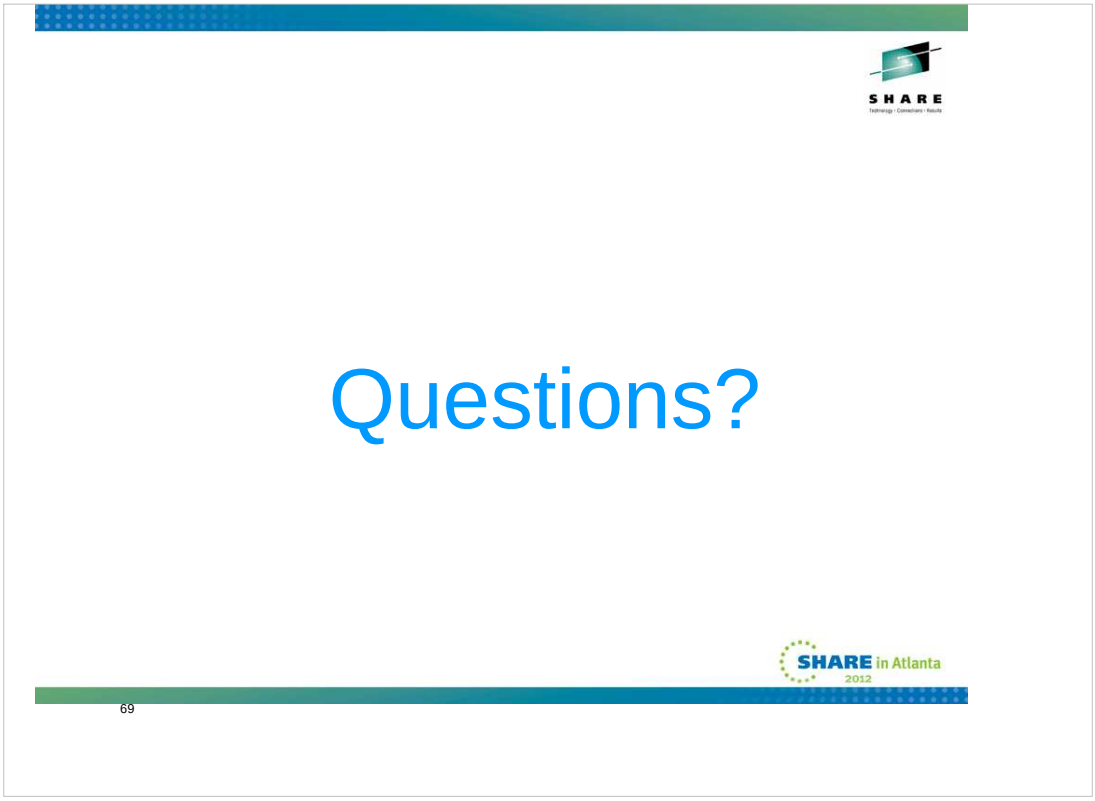
To:

```
#com.ibm.ws.scripting.connectionType=SOAP
com.ibm.ws.scripting.connectionType=RMI
com.ibm.ws.scripting.port=50003      /* to ORB port */
com.ibm.ws.scripting.defaultLang=jython
```

Use of RMI Connector with wsadmin.sh

- Now you can:

```
cd /wasv8config/s8cell/s8dmnode/DeploymentManager/profiles/default/bin
./wsadmin.sh
```



If you have any questions, now is the time...