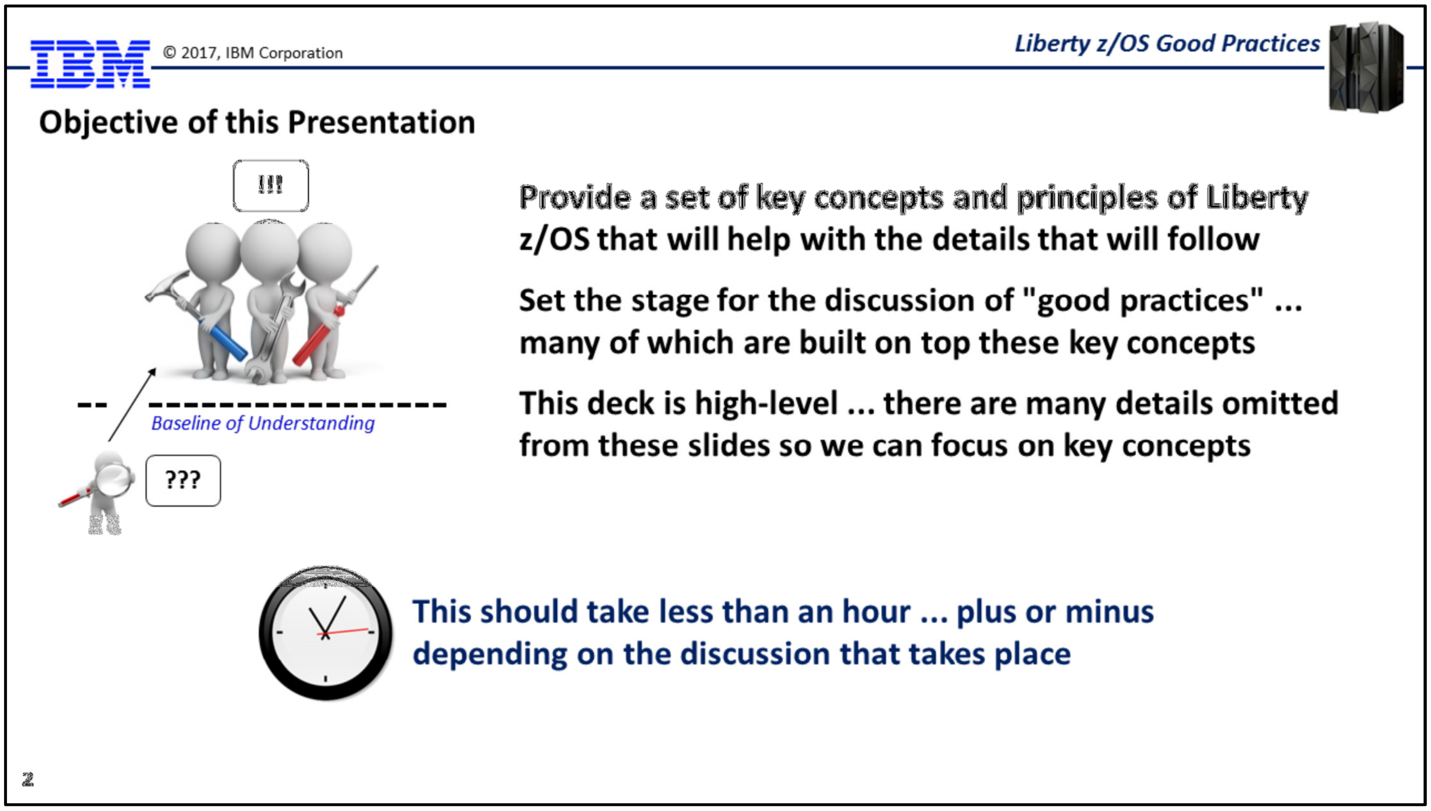
WebSphere Liberty z/OS

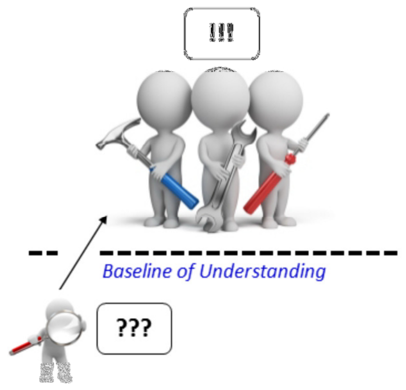
A review of key concepts



© 2017, IBM Corporation

Liberty z/OS Good Practices

Objective of this Presentation



Provide a set of key concepts and principles of Liberty z/OS that will help with the details that will follow

Set the stage for the discussion of "good practices" ... many of which are built on top these key concepts

This deck is high-level ... there are many details omitted from these slides so we can focus on key concepts

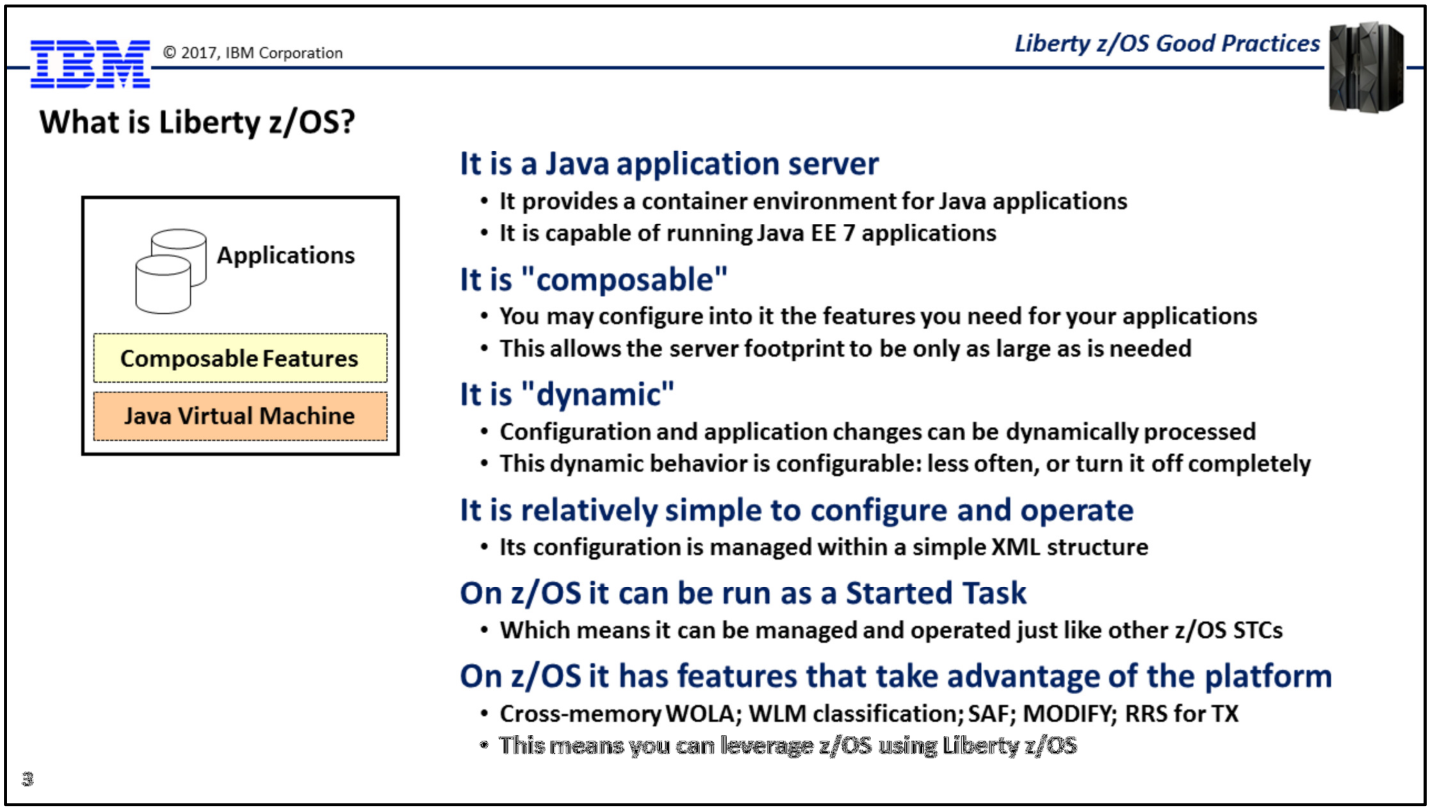
This should take less than an hour ... plus or minus depending on the discussion that takes place

2

Liberty, and Liberty on z/OS, may be a topic you're not familiar with. When a room full of people gather to discuss Liberty, there's a good chance the level of understanding in the room will be mixed. This presentation was designed to help "level set" attendees to some of the key concepts of Liberty z/OS.

As a "key concepts" presentation, many details will necessarily be left out. The focus will be on establishing a handful of core understandings about Liberty z/OS. Later, when we get into the specific-topic units, we can bring out more and more details.

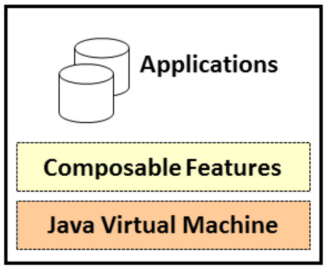
The intent of his presentation is to present these key concepts of Liberty in as "matter of fact" manner as possible.



© 2017, IBM Corporation

Liberty z/OS Good Practices

What is Liberty z/OS?



It is a Java application server

- It provides a container environment for Java applications
- It is capable of running Java EE 7 applications

It is "composable"

- You may configure into it the features you need for your applications
- This allows the server footprint to be only as large as is needed

It is "dynamic"

- Configuration and application changes can be dynamically processed
- This dynamic behavior is configurable: less often, or turn it off completely

It is relatively simple to configure and operate

- Its configuration is managed within a simple XML structure

On z/OS it can be run as a Started Task

- Which means it can be managed and operated just like other z/OS STCs

On z/OS it has features that take advantage of the platform

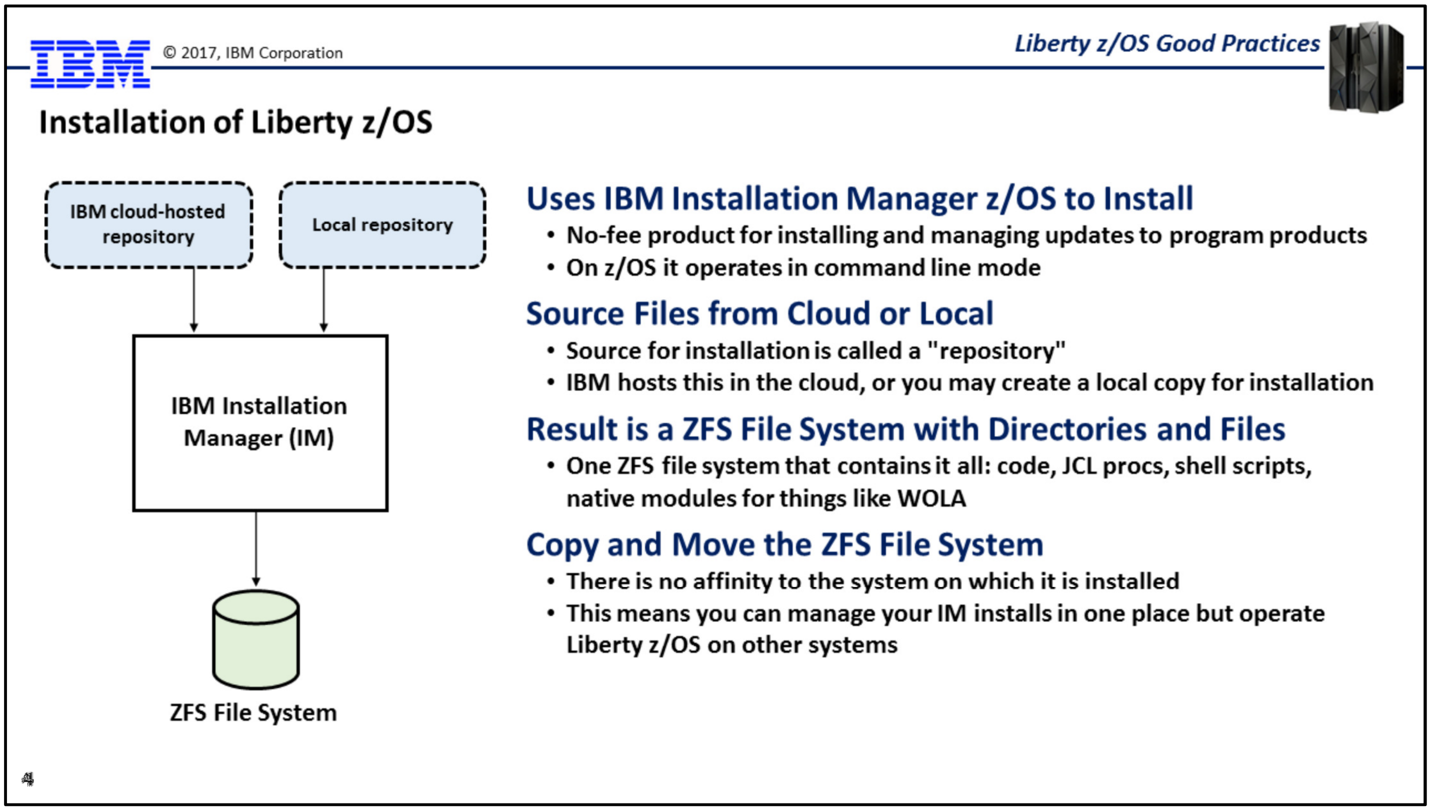
- Cross-memory WOLA; WLM classification; SAF; MODIFY; RRS for TX
- This means you can leverage z/OS using Liberty z/OS

3

We start with a discussion of Liberty z/OS and what that provides. This is the logical high-level starting point, and it also serves as a way to differential Liberty z/OS from WebSphere z/OS traditional, the previous "WebSphere Application Server" product. By the way, Liberty is available on many different platforms, not just z/OS, and its programming interfaces are identically across the different platforms.

- **Java Application Server** -- Liberty is a full Java EE application server, and provides a container-managed environment just like previous WAS servers. Application development is based on Java and Java EE concepts.
- **Composable** -- what's meant by this is that the features loaded by a given Liberty server are based on what you define to it. The design principle here is a given server can be tailored to load just those functions the applications require, and not more. This results in a smaller memory footprint. When operating hundreds (or thousands) of Liberty servers, the memory footprint savings can add up.
- **Dynamic** -- Liberty has at its heart the idea of detection of application and configuration changes, and dynamically loading those changes into the runtime. This avoids server stops and starts to pick up changes. In some environments dynamic updates may not be desired, and in those cases the rate of detection can be reduced, or the dynamic update capability can be turned off altogether.
- **Simple** -- this is really a statement of how Liberty compares to WAS traditional, which had dozens of configuration files in different locations. Liberty is based on the principle of a single configuration file. Later we will explore how side files can be "included" automatically, which allows you to modularize Liberty configuration if you wish.
- **Started Task** -- on z/OS the common operational model is to run Liberty as a started task. It is possible to run Liberty as a UNIX process, started from the shell, and you may wish to do that for development environments. But for production the expectation is Liberty will run as a started task, and Liberty z/OS supports this.
- **z/OS exploitation** -- work is run on z/OS in part because or the service attributes of the platform, so it follows that if something runs on z/OS it would be good to take advantage of those platform service attributes. Liberty z/OS does take advantage of several key z/OS service attributes. Later, when we get into discussing security and access to z/OS authorized services, you will see how a given server can be allowed to exploit the platform.

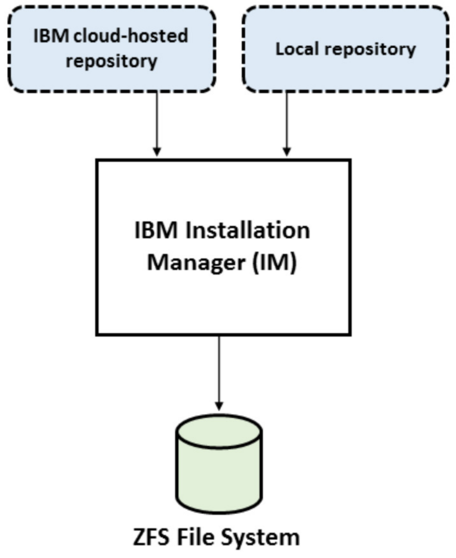
That is the high-level story in bullet format. As mentioned, many details are left out of this at this point. But if you take what's stated here on this chart and visualize a z/OS started task hosting a Java Virtual Machine and a Java EE framework, with a relatively simple configuration structure and dynamic updates, you have a picture of Liberty z/OS.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Installation of Liberty z/OS



Uses IBM Installation Manager z/OS to Install

- No-fee product for installing and managing updates to program products
- On z/OS it operates in command line mode

Source Files from Cloud or Local

- Source for installation is called a "repository"
- IBM hosts this in the cloud, or you may create a local copy for installation

Result is a ZFS File System with Directories and Files

- One ZFS file system that contains it all: code, JCL procs, shell scripts, native modules for things like WOLA

Copy and Move the ZFS File System

- There is no affinity to the system on which it is installed
- This means you can manage your IM installs in one place but operate Liberty z/OS on other systems

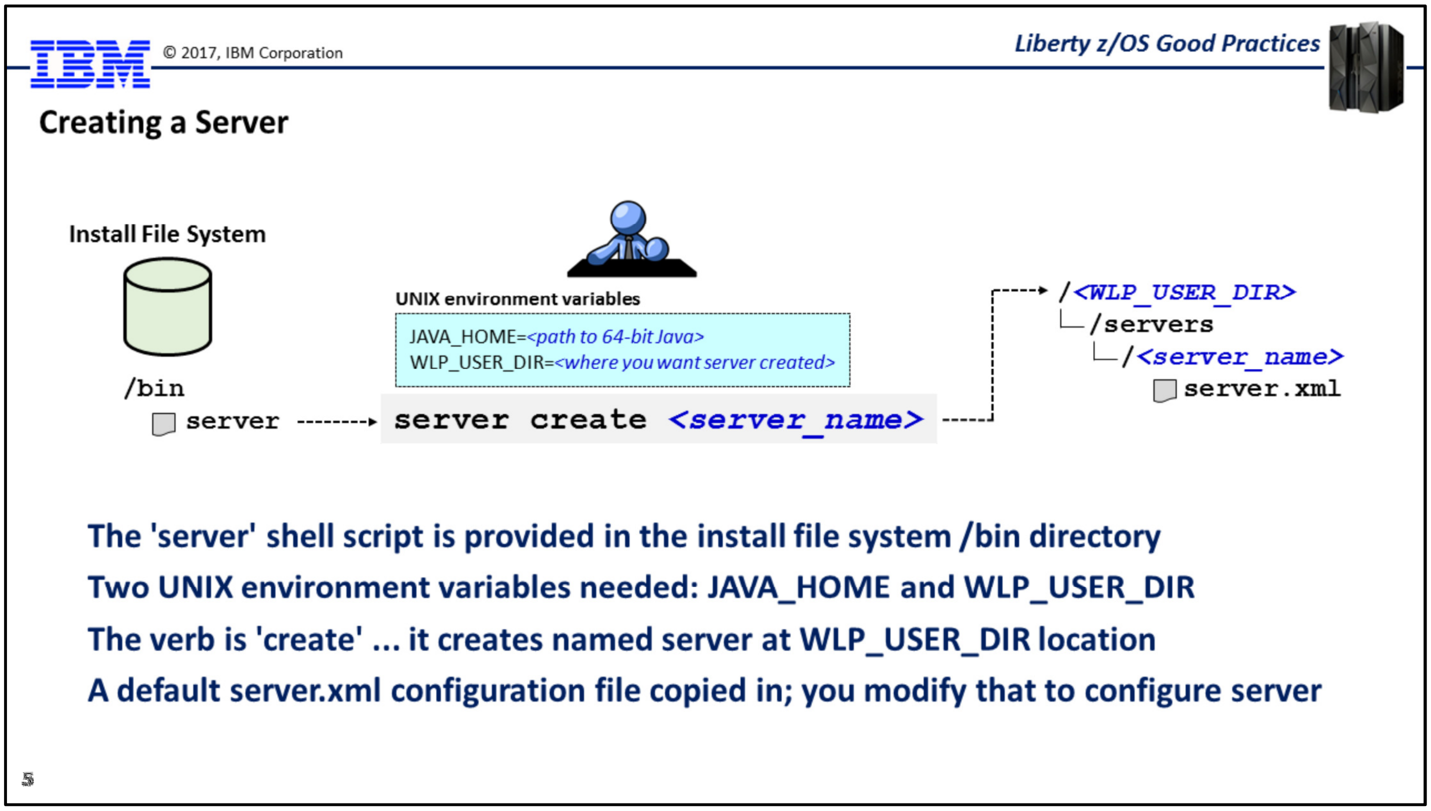
4

The installation of Liberty z/OS is performed using IBM Installation Manager (IM) and *not* SMP/E. IM on z/OS is available without a license charge, and once IM itself is installed you can use it to install other IM-installable software. IM operates on z/OS with a command line interface, and the commands you enter can be wrappered with JCL so the installation can be made easily repeatable.

Like any installation utility, IM requires a source of files for performing the installation. This is called the "repository," and in reality it is little more than a large ZIP file created by IBM. The two basic choices are to download the repository and use it "locally," or you can have IM access "the cloud" (meaning: an IBM-hosted server where the repository resides) and install from a remote repository. More and more are using the remote "cloud based" install because it avoids the extra step of downloading and using your own DASD to hold the source repository ZIP.

The result of a Liberty z/OS installation using IM is a file system, most likely a ZFS file system nowadays (as opposed to the older HFS file system design). The key point is that's the install -- UNIX file system. All the components are held there: the native module files, the JCL start procs, the sample XML files. There is no affinity to the system you install on, so the idea of a "service zone" is quite applicable to IM installations: you install on LPAR X, which is isolated from others, and then you copy the ZFS file system to the environments where Liberty z/OS will be used.

Generally speaking the installation is easy. As stated, once you have IM itself installed and the installation command syntax working, it becomes a fairly rote exercise for future installations. There are good practices around maintaining copies of installations by version and fixpack level, and we'll discuss those in the other units.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Creating a Server

Install File System
/bin
server
UNIX environment variables
JAVA_HOME=<path to 64-bit Java>
WLP_USER_DIR=<where you want server created>
server create <server_name>
/<WLP_USER_DIR>
/servers
/<server_name>
server.xml

- The 'server' shell script is provided in the install file system /bin directory
- Two UNIX environment variables needed: JAVA_HOME and WLP_USER_DIR
- The verb is 'create' ... it creates named server at WLP_USER_DIR location
- A default server.xml configuration file copied in; you modify that to configure server

5

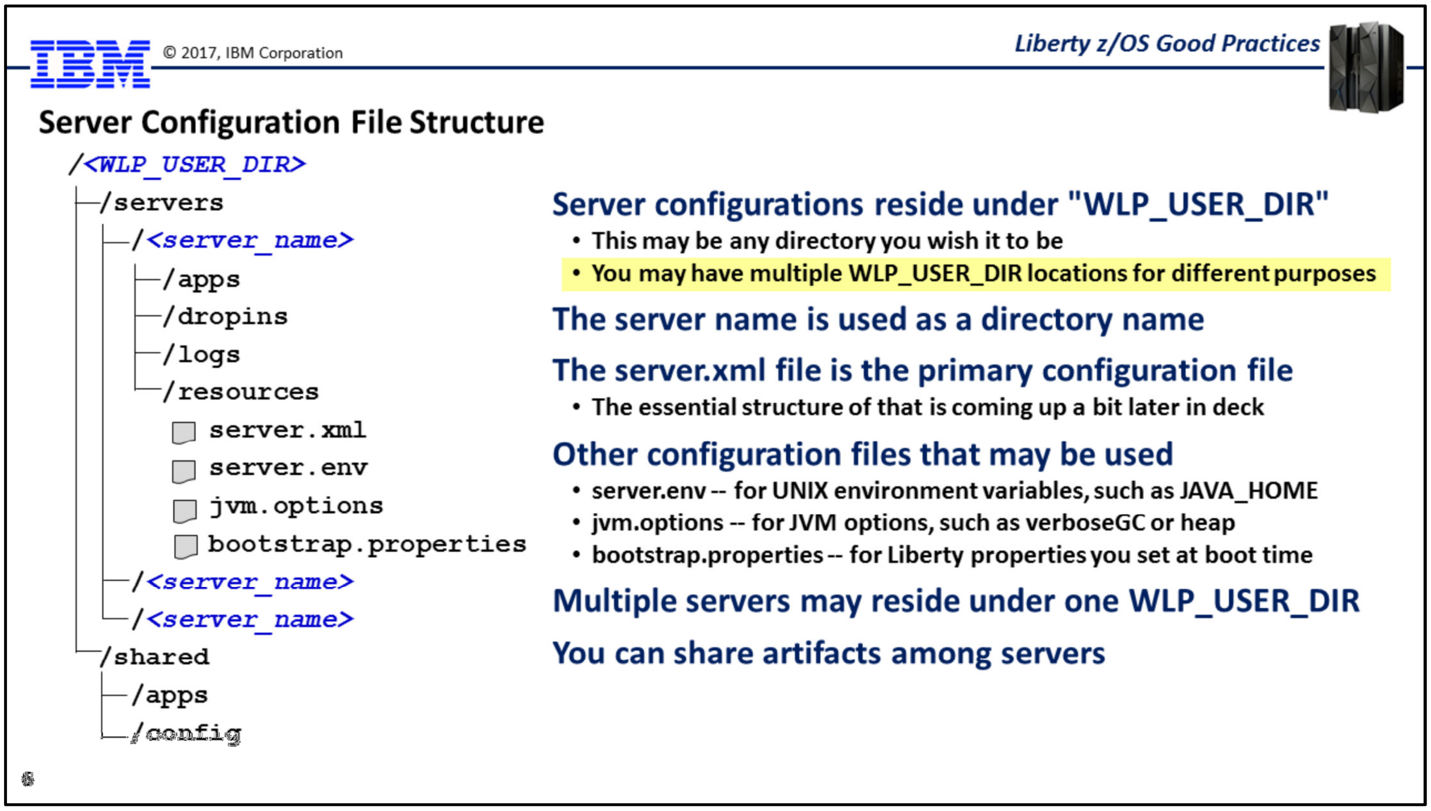
The process of creating a server configuration in Liberty is fairly simple: it involves using a supplied shell script (located under the install location /bin directory, and called 'server'). The server shell script has several action verbs associated with it, one of which is "create".

The server shell script is invoked in a z/OS UNIX shell. You can access the shell in several ways: by using a telnet or SSH client and connecting to the z/OS server, or by using OMVS (a 3270 shell environment), or by using BPXBATCH inside of JCL to create a shell and issue a command. Two UNIX environment variables are needed to make this work -- (1) you need to tell the shell environment where a valid 64-bit Java resides, which you do with the JAVA_HOME variable; and (2) you need to tell the shell environment where you want the server to be created, which you do with the WLP_USER_DIR variable. The WLP_USER_DIR variable can be any location you want. The only requirement is the ID under which you run the server shell script has to have write permissions.

Note: later we will have a great deal of discussion about the "WLP_USER_DIR" because quite a few other good practices flow out of that, such as isolation between environments, sharing applications and configuration elements between servers, and UNIX file permission security practices. For now the key point is that output goes to wherever you specify with the UNIX environment variable WLP_USER_DIR.

The server name you supply to the shell script becomes the UNIX directory under the /servers directory. The server name can be any name that's valid for a UNIX directory name. As we get into more of the details you'll see that there are some good practices with respect to naming conventions, and aligning the server name with other z/OS elements such as user ID values and other SAF security profiles. The message here is while the server name *can* be pretty much anything you want, you'll likely want to plan out and control the server names. The units that follow will help you do that.

A default configuration file -- server.xml -- will be copied in from the installation location. You can, if you want, create other "template" server.xml files and have them copied in when the server create is issued. When first starting out you will likely allow the default template to come in, and you'll modify that.



Liberty z/OS Good Practices

Server Configuration File Structure

```
/<WLP_USER_DIR>
 ├─/servers
 │  ├─/<server_name>
 │  │  ├─/apps
 │  │  ├─/dropins
 │  │  ├─/logs
 │  │  └─/resources
 │  │     server.xml
 │  │     server.env
 │  │     jvm.options
 │  │     bootstrap.properties
 │  ├─/<server_name>
 │  └─/<server_name>
 └─/shared
    ├─/apps
    └─/config
```

Server configurations reside under "WLP_USER_DIR"
- This may be any directory you wish it to be
- You may have multiple WLP_USER_DIR locations for different purposes

The server name is used as a directory name

The server.xml file is the primary configuration file
- The essential structure of that is coming up a bit later in deck

Other configuration files that may be used
- server.env -- for UNIX environment variables, such as JAVA_HOME
- jvm.options -- for JVM options, such as verboseGC or heap
- bootstrap.properties -- for Liberty properties you set at boot time

Multiple servers may reside under one WLP_USER_DIR

You can share artifacts among servers

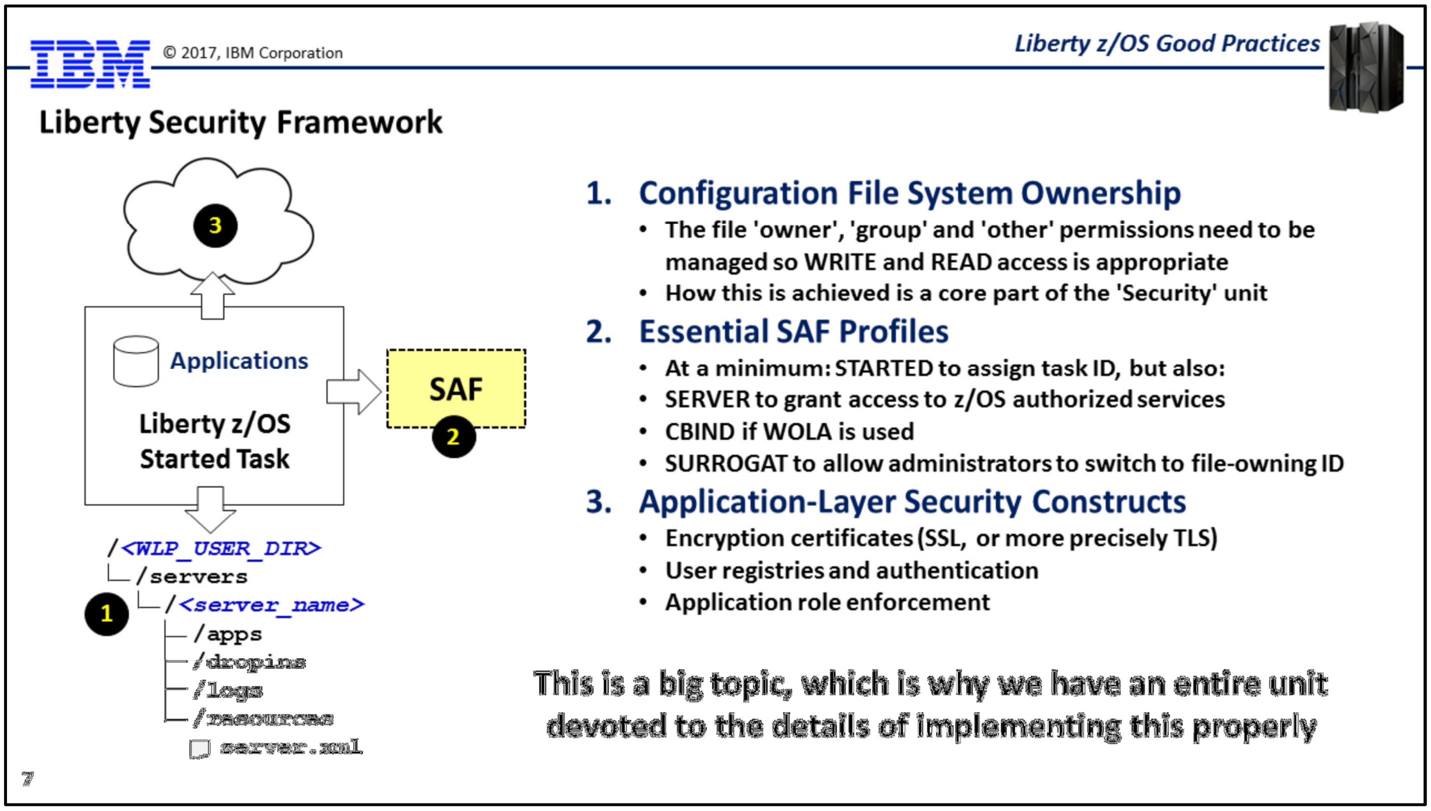
6

The picture in this chart illustrates a 'typical' directory and file structure under a WLP_USER DIR. It's important to note that you may have multiple WLP_USER_DIR locations, not just one. The reason you may want multiple has to do with separating operational environments, and UNIX file permission security. We'll get to that later.

As mentioned on the earlier chart, the server name you provided with the 'server create' command is used for the UNIX directory under which directories and files for that server reside. The server.xml file is the primary configuration file, but you may add others -- server.env (UNIX environment), jvm.options (to control JVM settings like heap, or verbose GC recording), and bootstrap.properties (to supply properties that take effect before the JVM is instantiated. The server.xml file is the key configuration file.

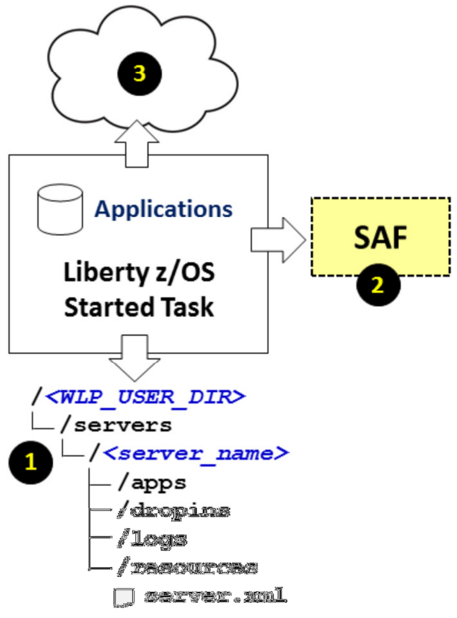
You may have multiple servers defined under a given WLP_USER_DIR location. In fact, there's no Liberty architectural limit to how many servers you can define under a given WLP_USER_DIR location. Servers that reside under a given WLP_USER_DIR location have the ability to share applications and configuration elements rather easily by accessing those shared components housed down under the /shared directory. Servers under the same WLP_USER_DIR may also be started with a common JCL start procedure, with the server name being passed in with a PARMS= value.

So this becomes one of the early considerations in planning for Liberty z/OS -- how will you use different WLP_USER_DIR locations to effect server grouping so the isolation between groups is what you desire? There is quite a bit of discussion that can go around this topic, and we'll see that come out as we work through the other units.



Liberty z/OS Good Practices

Liberty Security Framework



1. **Configuration File System Ownership**
   - The file 'owner', 'group' and 'other' permissions need to be managed so WRITE and READ access is appropriate
   - How this is achieved is a core part of the 'Security' unit
2. **Essential SAF Profiles**
   - At a minimum: STARTED to assign task ID, but also:
   - SERVER to grant access to z/OS authorized services
   - CBIND if WOLA is used
   - SURROGAT to allow administrators to switch to file-owning ID
3. **Application-Layer Security Constructs**
   - Encryption certificates (SSL, or more precisely TLS)
   - User registries and authentication
   - Application role enforcement

This is a big topic, which is why we have an entire unit devoted to the details of implementing this properly

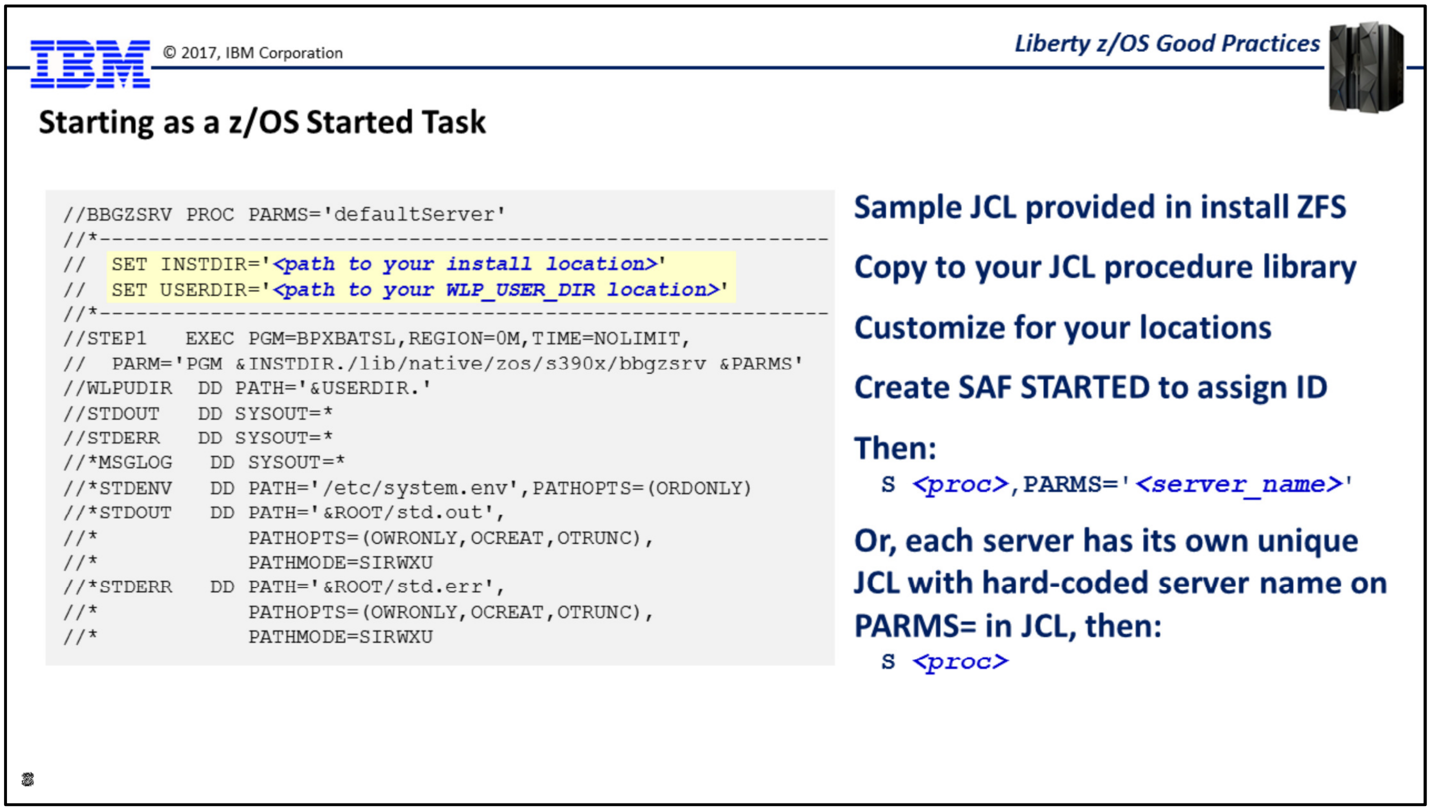
7

Security is a big topic, and providing all the details in one chart is impossible. But what we *can* do is present a framework of security topics. That framework is helpful in organizing the conversation around security.

There are three essential areas of security discussions around Liberty z/OS, and they are illustrated in the picture above starting at the bottom and working to the top:

1. **UNIX file system permissions** -- this is standard UNIX file permissions, but it plays a big role in how you control who can gain access to server configuration files for write, or even to look at configuration files. There are also considerations to keep in mind if you want the started task ID to be different from the file owning ID. This is all part of the security unit.
2. **SAF profiles** -- there are a handful of SAF profiles that come into play with Liberty z/OS: STARTED (for assigning the ID to the started task); SERVER (for granting access to z/OS authorized services through the Liberty Angel process); and CBIND (for controlling who can use a key WebSphere Optimized Local Adapters function); and potentially others, depending on how you design things (SURROGAT for granting ability to 'su' to an ID, EJBROLE for application roles, KEYRING for holding digital certificates). It is possible to avoid all these SAF definitions by running your server as a UNIX process, and limiting what the server does. But the more we talk about Liberty z/OS in a production setting, the more SAF will become part of the discussion.
3. **Application security** -- at this layer we move into the realm of things like encryption, authentication and authorization. These application-layer security constructs are common across platforms on which Liberty runs, though it is possible to use SAF for some of this under the application API layer. (For instance, you can have an application that defines a role, and the role checking can be done using SAF EJBROLE definitions. But the application does not know SAF is involved; that is a function of Liberty z/OS checking the role and going to SAF because a configuration element in the server.xml tells that server instance of use SAF.)

Our reason for crafting this three-level framework is it allows us to keep the security discussion focused within an area, and not make the topic -- already somewhat confusing -- more so by mixing different areas.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Starting as a z/OS Started Task

```
//BBGZSRV PROC PARMS='defaultServer'
//*-------------------------------------------------------------
//  SET INSTDIR='<path to your install location>'
//  SET USERDIR='<path to your WLP_USER_DIR location>'
//*-------------------------------------------------------------
//STEP1   EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
//  PARM='PGM &INSTDIR./lib/native/zos/s390x/bbgzsrv &PARMS'
//WLPUDIR  DD PATH='&USERDIR.'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//*MSGLOG   DD SYSOUT=*
//*STDENV   DD PATH='/etc/system.env',PATHOPTS=(ORDONLY)
//*STDOUT   DD PATH='&ROOT/std.out',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//*STDERR   DD PATH='&ROOT/std.err',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
```

Sample JCL provided in install ZFS

Copy to your JCL procedure library

Customize for your locations

Create SAF STARTED to assign ID

Then:

```
S <proc>,PARMS='<server_name>'
```

Or, each server has its own unique JCL with hard-coded server name on PARMS= in JCL, then:

```
S <proc>
```

As noted earlier, you can start a Liberty z/OS server as a z/OS started task (STC). The Liberty z/OS installation includes a sample JCL member which you can copy out of the file system to your procedure library and modify.

The Liberty z/OS started task requires knowledge of two key things -- where Liberty z/OS is installed, and what WLP_USER_DIR to look under. Those are supplied with two 'SET' variables in the JCL start procedure. The JCL proc is also designed to take as an input parameter the server name you wish to start. So the START command looks something like this:

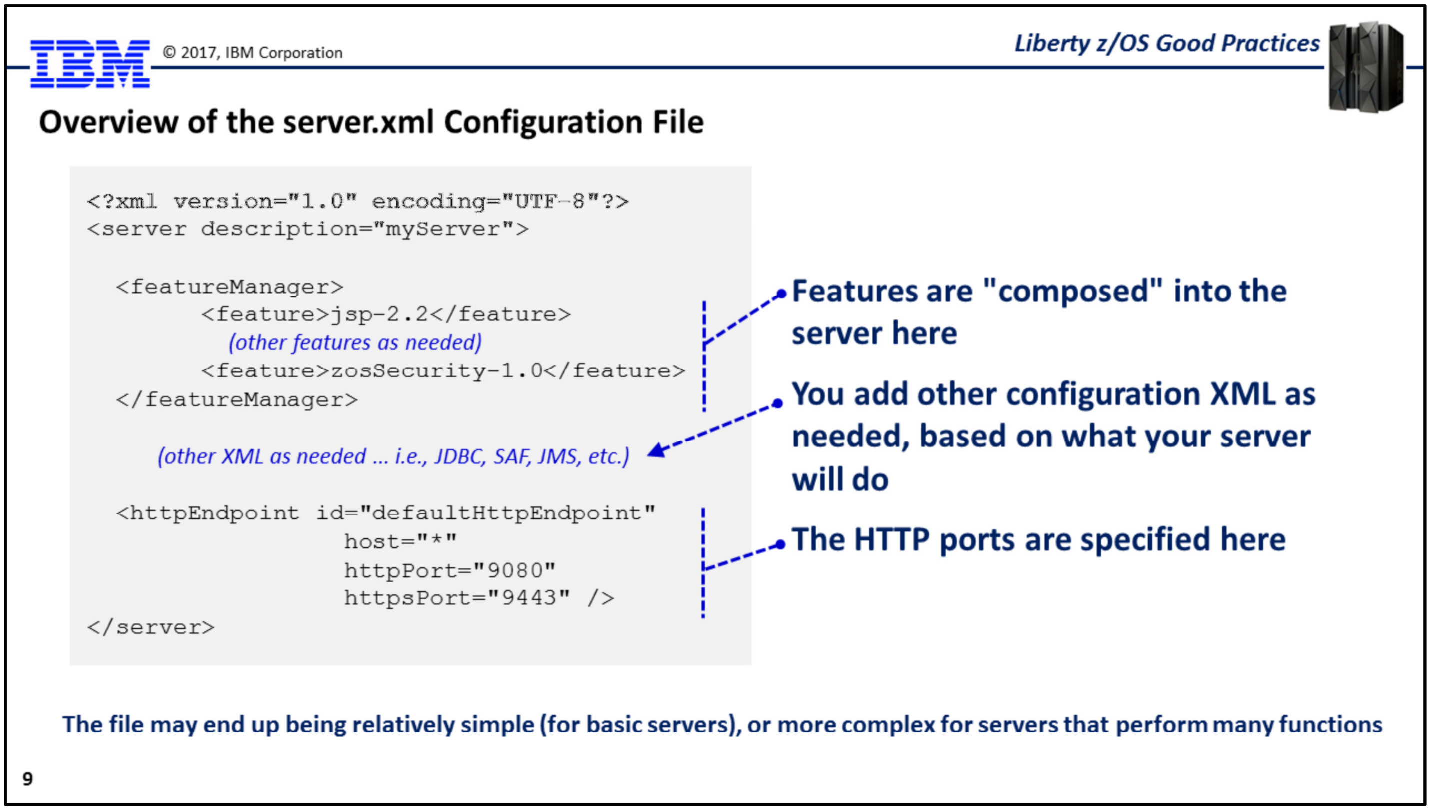
```
START BBGZSRV,PARMS='myServer'
```

You may, if you wish, supply a JOBNAME=:

```
START BBGZSRV,JOBNAME=MYSERVER,PARMS='myServer'
```

This JCL can be tailored based on your needs. For example, you may rename the JCL start procedure to match your own naming standards. You can hard-code the server name on the PROC statement where there's a default PARMS= value supplied. You can go further still and customize other parameters you pass in on the START command that are used to resolve the install path or WLP_USER_DIR location. There is considerable flexibility. When it comes to the parameters you pass in you do need to be careful not to exceed the maximum parameter length (100 characters).

For someone familiar with z/OS started tasks, this should look familiar. There is nothing special or fancy going on here: this is relatively simple JCL that launches the Liberty server you specify.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Overview of the server.xml Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="myServer">

  <featureManager>
        <feature>jsp-2.2</feature>
          (other features as needed)
        <feature>zosSecurity-1.0</feature>
  </featureManager>

     (other XML as needed ... i.e., JDBC, SAF, JMS, etc.)

  <httpEndpoint id="defaultHttpEndpoint"
                host="*"
                httpPort="9080"
                httpsPort="9443" />
</server>
```

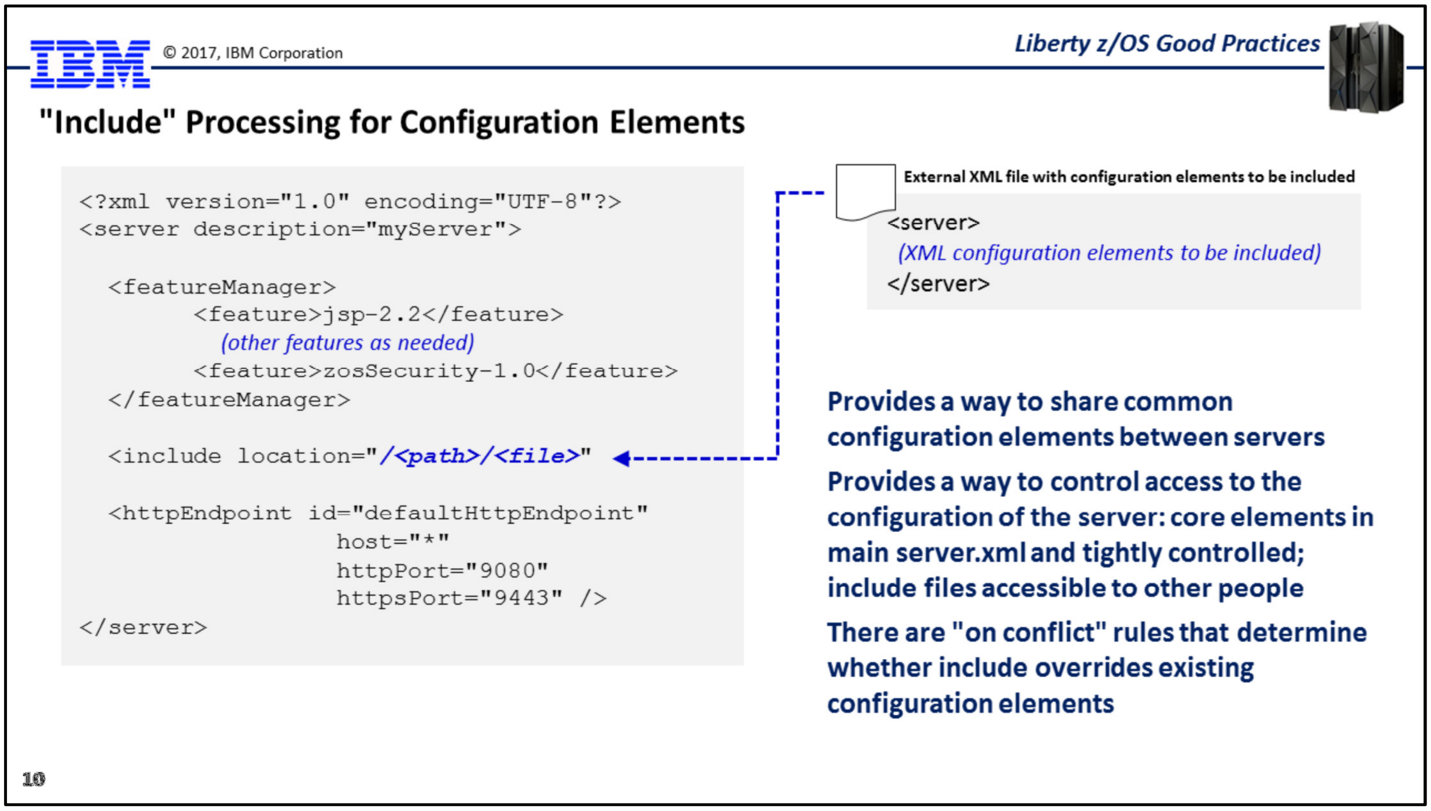
Features are "composed" into the server here
You add other configuration XML as needed, based on what your server will do
The HTTP ports are specified here

The file may end up being relatively simple (for basic servers), or more complex for servers that perform many functions

9

As mentioned, the server.xml file is the key configuration file for a Liberty server. That's true for Liberty on distributed and Liberty z/OS. The structure of this server.xml can be very simple, or somewhat complex. It all depends on what you are seeking to do with the server.

The picture above illustrates the server.xml file and provides some guidance to understanding some of the key elements of the file. For example, the "features" are what you provide to tell the server what functions to load. The HTTP port values are coded in the <httpEndpoint> section. And between can be all manner of other XML -- well documented in the online Knowledge Center -- to do things such as configure JDBC drivers, or MQ definitions, or WOAL definitions, or whatever.


© 2017, IBM Corporation
Liberty z/OS Good Practices

"Include" Processing for Configuration Elements

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="myServer">

  <featureManager>
        <feature>jsp-2.2</feature>
          (other features as needed)
        <feature>zosSecurity-1.0</feature>
  </featureManager>

  <include location="/<path>/<file>"

  <httpEndpoint id="defaultHttpEndpoint"
                host="*"
                httpPort="9080"
                httpsPort="9443" />
</server>
```

External XML file with configuration elements to be included

```
<server>
 (XML configuration elements to be included)
</server>
```

Provides a way to share common configuration elements between servers

Provides a way to control access to the configuration of the server: core elements in main server.xml and tightly controlled; include files accessible to other people

There are "on conflict" rules that determine whether include overrides existing configuration elements

10

You could maintain all the XML elements for a server within one file, or you may "include" portions from other files. There are several reasons why you may wish to use this "include" function:

- You have configuration elements that are common to many servers, and rather than duplicating the same XML over and over again, you may create one copy and point to it using <include> from the other servers.
- You have configuration elements you wish to more tightly control, such as JDBC definitions, that you maintain in side files. The UNIX 'write' permissions on those files are restricted so only a select few can change them. But the 'read' permissions allow other servers to include the XML elements.
- You have configuration elements that are somewhat sensitive in nature, such as encoded password strings you'd prefer others to not be able to see. Here again, you lock down the 'read' permissions so *only* the server IDs can read them, but others can not.

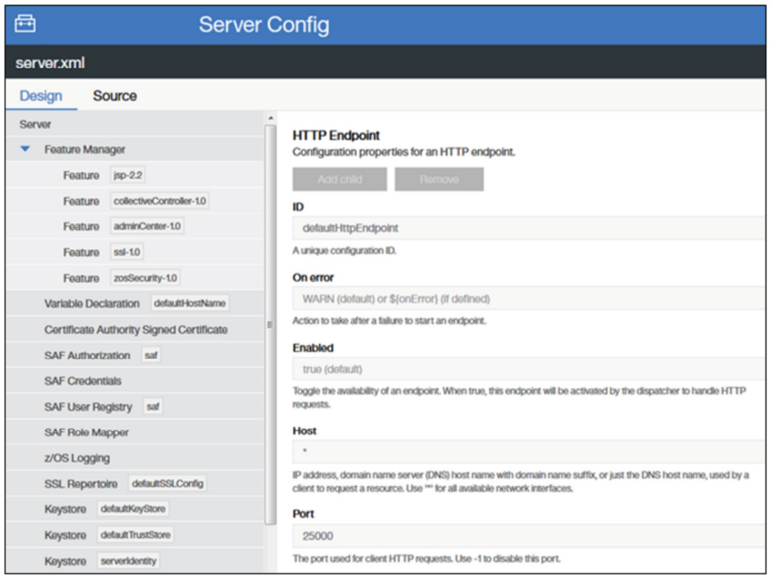
You may have other reasons to do this beyond the three examples cited here.

You may wonder what happens when a configuration element is in the main server.xml and it's in the XML brought in with an <include> statement. There are "on conflict" rules you can configure on the <include> statement which control what happens when such an event occurs. So you can override using the included content, or you can discard the included content.

We introduced this concept of <include> in this "key concepts" deck because we anticipate this feature will be widely used when Liberty z/OS is use in a production environment where multiple servers are involved. The benefits to separating out common or sensitive configuration elements is compelling.

Liberty z/OS Good Practices

AdminCenter Server Configuration Wizard


Server Config

server.xml

Design Source

Server
- Feature Manager
  - Feature jsp-2.2
  - Feature collectiveController-1.0
  - Feature adminCenter-1.0
  - Feature ssl-1.0
  - Feature zosSecurity-1.0
- Variable Declaration defaultHostName
- Certificate Authority Signed Certificate
- SAF Authorization saf
- SAF Credentials
- SAF User Registry saf
- SAF Role Mapper
- z/OS Logging
- SSL Repertoire defaultSSLConfig
- Keystore defaultKeyStore
- Keystore defaultTrustStore
- Keystore serverIdentity

HTTP Endpoint
Configuration properties for an HTTP endpoint.

Add child Remove

ID
defaultHttpEndpoint
A unique configuration ID.

On error
WARN (default) or ${onError} (if defined)
Action to take after a failure to start an endpoint.

Enabled
true (default)
Toggle the availability of an endpoint. When true, this endpoint will be activated by the dispatcher to handle HTTP requests.

Host
*
IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used by a client to request a resource. Use "*" for all available network interfaces.

Port
25000
The port used for client HTTP requests. Use -1 to disable this port.

The AdminCenter is a feature that can be added to a server

It provides a browser-based graphical interface

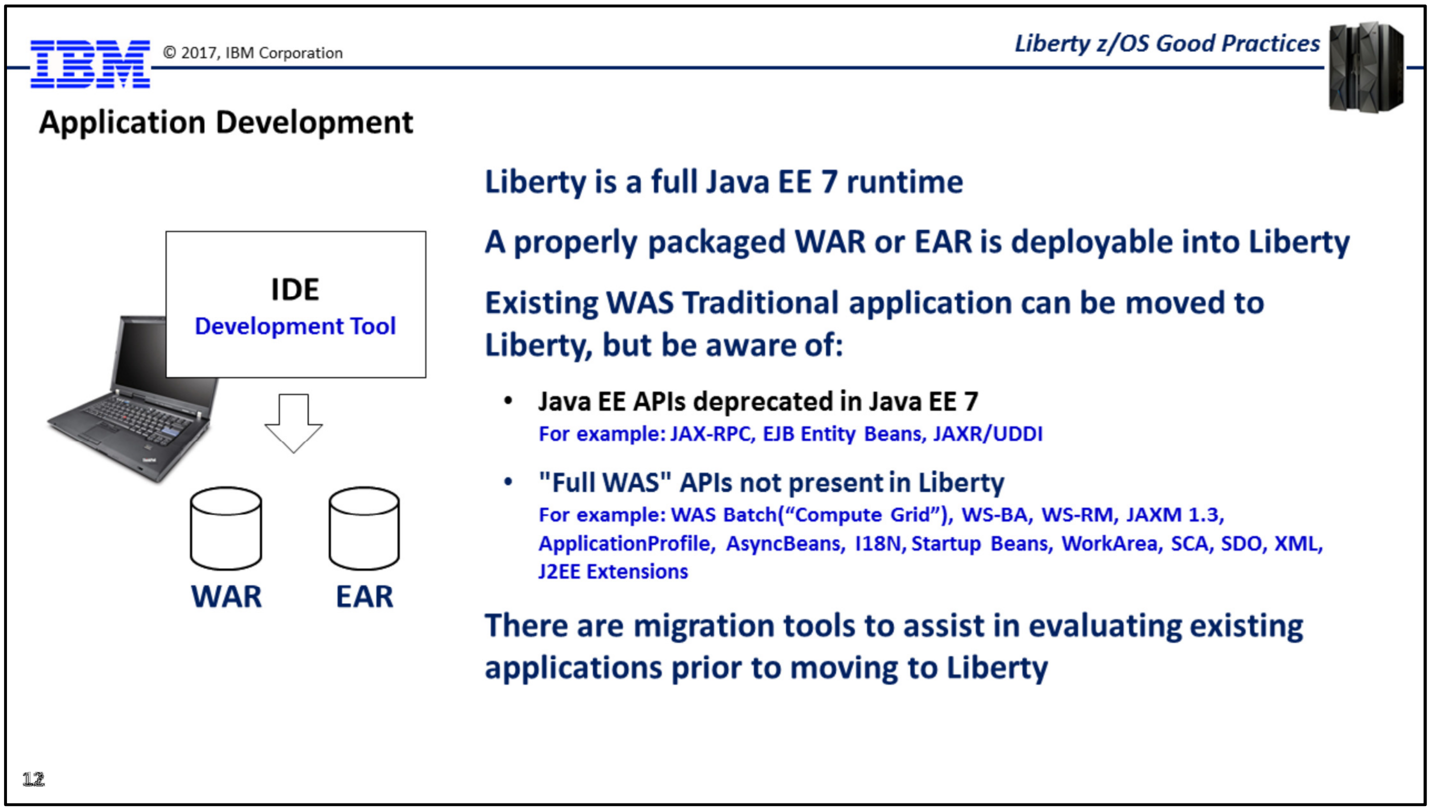
The 'Server Config' tool provides a way to view and modify the server.xml using a configuration wizard
- **Provided the Admin ID has write access to the server.xml**

Or, if you prefer, you can add XML directly to the server.xml

11

Liberty has a graphical management interface called the "AdminCenter" which can provide some assistance with configuring the server.xml file. The AdminCenter is a "feature" you configure into the server.xml, and with the AdminCenter function is loaded and available. One aspect of the AdminCenter is a server configuration wizard which will read in the server.xml for the server and provide you the ability to add other elements using the wizard.

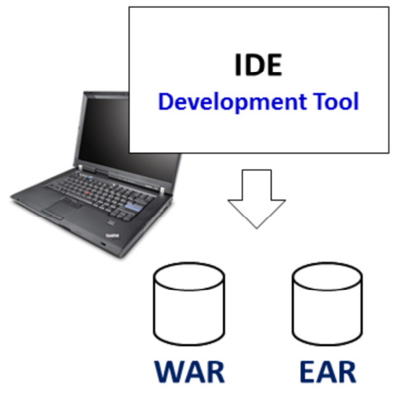
It's a handy tool. We anticipate as you get more familiar with Liberty configuration, you will do more manual updates based on other working samples you have. But this tool can be helpful in guiding you to the XML elements you require the first time you are configuring up a new function.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Application Development



Liberty is a full Java EE 7 runtime

A properly packaged WAR or EAR is deployable into Liberty

Existing WAS Traditional application can be moved to Liberty, but be aware of:

- Java EE APIs deprecated in Java EE 7
  For example: JAX-RPC, EJB Entity Beans, JAXR/UDDI
- "Full WAS" APIs not present in Liberty
  For example: WAS Batch("Compute Grid"), WS-BA, WS-RM, JAXM 1.3, ApplicationProfile, AsyncBeans, I18N, Startup Beans, WorkArea, SCA, SDO, XML, J2EE Extensions

There are migration tools to assist in evaluating existing applications prior to moving to Liberty

12

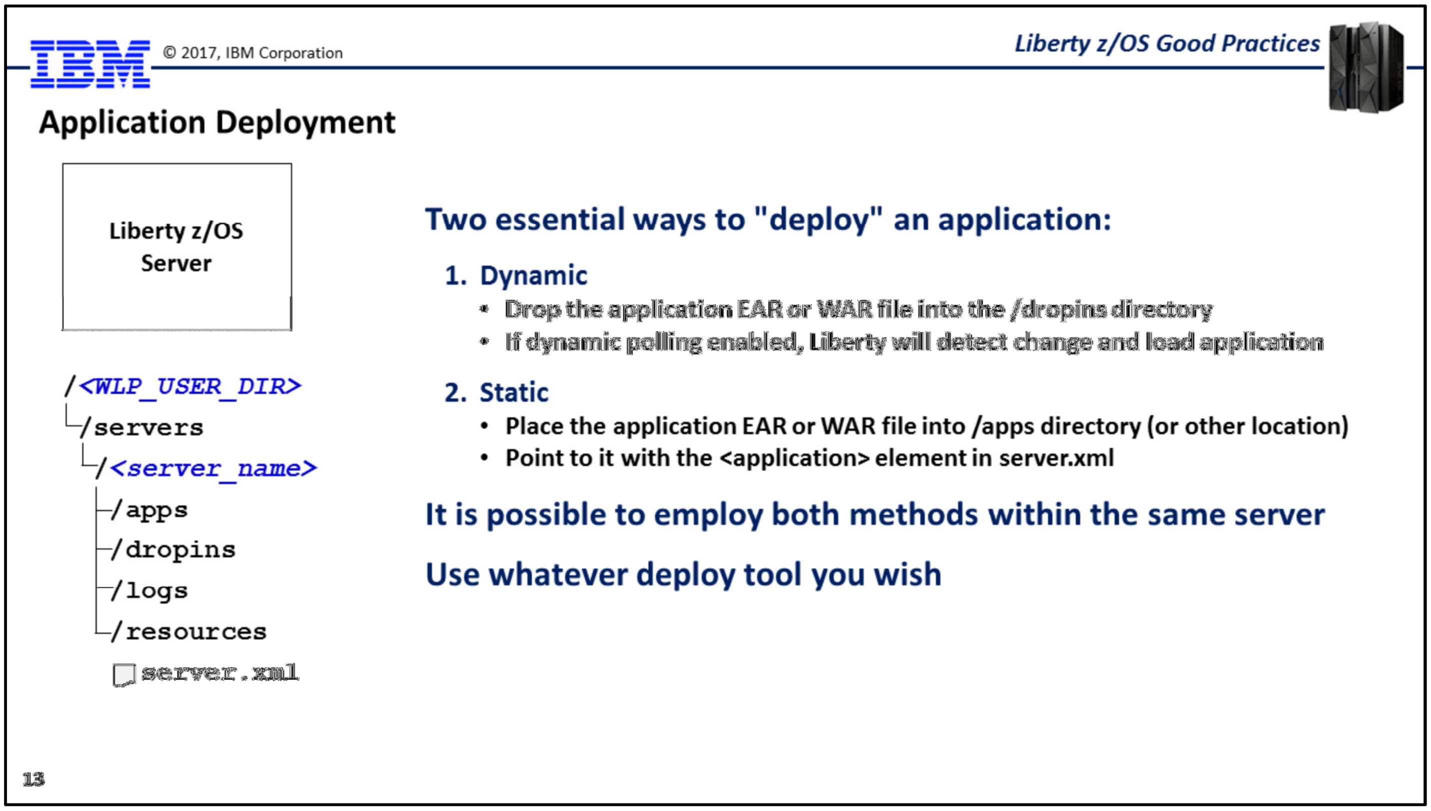
Application development for Liberty is the same as application development for any Java EE 7 server runtime. So there's nothing different or special about it. Any properly packaged WAR or EAR is deployable into a Liberty server, whether on distributed or z/OS.

Note: because Liberty is "composable," it implies the features the application seeks to use are also configured into the Liberty server. An application that looks to use JMS for message queuing will require the JMS feature to be loaded.

Where the conversation gets a bit more involved is when we discuss applications that were originally written for WAS traditional and are now to be moved to Liberty. Here we have to be a little careful to make sure the applications do not make use of APIs that are present in WAS traditional but not present in Liberty. At a high level, there are two things to concern ourselves with:

- **Java EE APIs** -- the Java EE specification changes over time, and certain APIs are *deprecated* (present, but marked for eventual removal) or *removed* (no longer in the more recent Java EE implementation). If the application to be moved from WAS traditional to Liberty makes use of these older Java EE APIs, then it may present an issue when they are moved to Liberty with its more recent Java EE 7 implementation.
- **"Full WAS," or WAS traditional APIs** -- WAS traditional had a set of APIs that went above and beyond those offered in the open standard Java EE specification. Many of those "above and beyond" APIs are not present in Liberty. The chart provides a list of some that are not present in Liberty. Here again, the question is what APIs the application being moved uses. If the application seeks APIs that are not present in Liberty, then the application will have issues.

IBM does have a set of tools to analyze applications and report on what interface considerations to investigate to insure a move from WAS traditional to Liberty. Later we'll provide the URL for those utilities. For now, the "key concept" here is that applications *can* be moved from WAS traditional to Liberty, provided the application is using APIs that are present in Liberty. If the application is using APIs that are in WAS traditional but not Liberty, then the application will need to be inspected and modified to run in Liberty.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Application Deployment

Liberty z/OS Server

```
/<WLP_USER_DIR>
└/servers
  └/<server_name>
    ├/apps
    ├/dropins
    ├/logs
    └/resources
      server.xml
```

Two essential ways to "deploy" an application:

1. **Dynamic**
   - Drop the application EAR or WAR file into the /dropins directory
   - If dynamic polling enabled, Liberty will detect change and load application
2. **Static**
   - Place the application EAR or WAR file into /apps directory (or other location)
   - Point to it with the <application> element in server.xml

It is possible to employ both methods within the same server

Use whatever deploy tool you wish

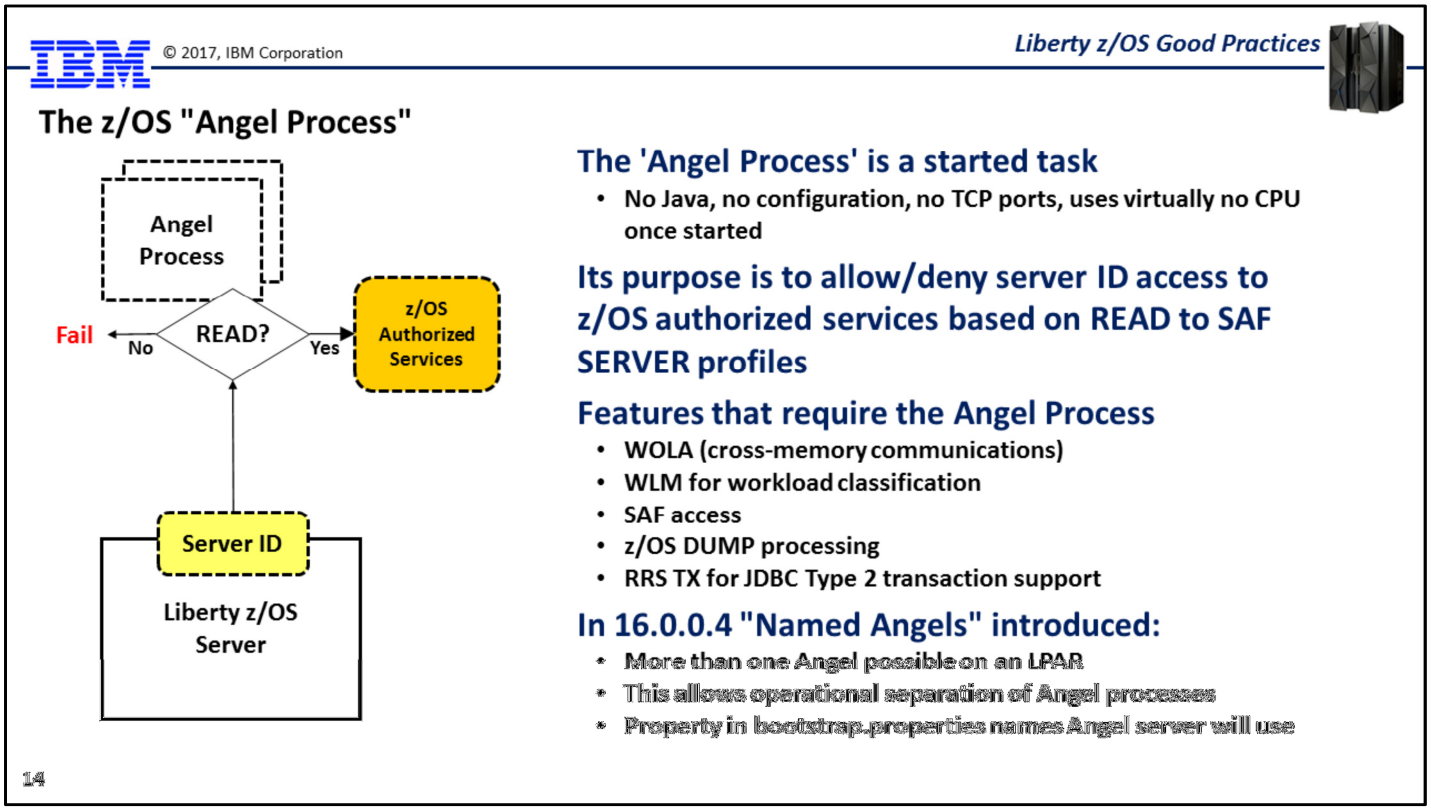
13

Deploying an application into Liberty is fairly simple. It is an easier process than with WAS traditional, which required the application deployment to be made through the management interface, either the Admin Console or the WSADMIN scripting interface. Not so with Liberty: with Liberty you only need to put the application file in an accessible location and let the server know about it. There are two general approaches:

- **Dynamic** -- here you take advantage of the dynamic model of Liberty and you simply "drop" the application file into the directory Liberty is monitoring. By default this is the /dropins directory, but you may define another location if you wish. When Liberty detects the new file is present -- either when the next polling interval is reached, or the management bean is invoked to tell Liberty to go look -- it will load up the application. There is no explicit definition of the application in the server.xml.
- **Static** -- here you make an explicit reference to the application from the server.xml and Liberty goes to that location and loads the application. This may actually be a "dynamic" load if you have dynamic *configuration* change detection enabled. In that case, Liberty will detect the change to server.xml and go load the application. If you have dynamic configuration change turned off, then the application will be loaded at the next server restart. We anticipate for production scenarios the dynamic nature of Liberty will be turned off, and all application updates will be managed through manual updates during maintenance change windows.

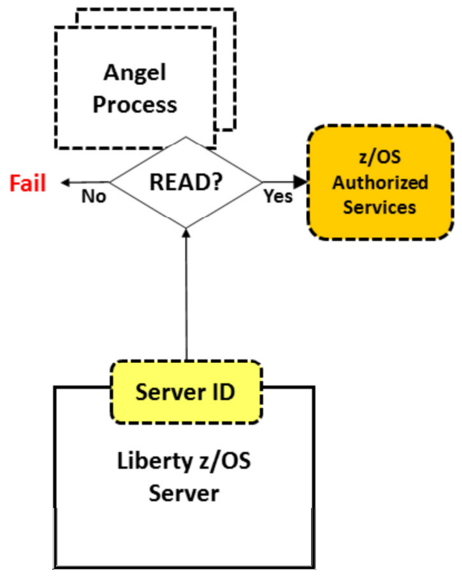
As the chart indicates, within a single application server you may take advantage of both the /dropins directory application deployment and an explicit pointer to the application with an <application> tag element.

As for deployment tools, you may use whatever deployment tool suits your needs. At a minimum that tool would simply upload a file to the /dropins directory. Or it could upload the file to file system location and modify the server.xml to include an <application> tag reference to the application. A more sophisticated approach would be to upload the file, change server.xml, then invoke the management bean to tell Liberty to go refresh its configuration knowledge and take the actions implied by the configuration changes.



Liberty z/OS Good Practices

The z/OS "Angel Process"



The 'Angel Process' is a started task

- No Java, no configuration, no TCP ports, uses virtually no CPU once started

Its purpose is to allow/deny server ID access to z/OS authorized services based on READ to SAF SERVER profiles

Features that require the Angel Process

- WOLA (cross-memory communications)
- WLM for workload classification
- SAF access
- z/OS DUMP processing
- RRS TX for JDBC Type 2 transaction support

In 16.0.0.4 "Named Angels" introduced:

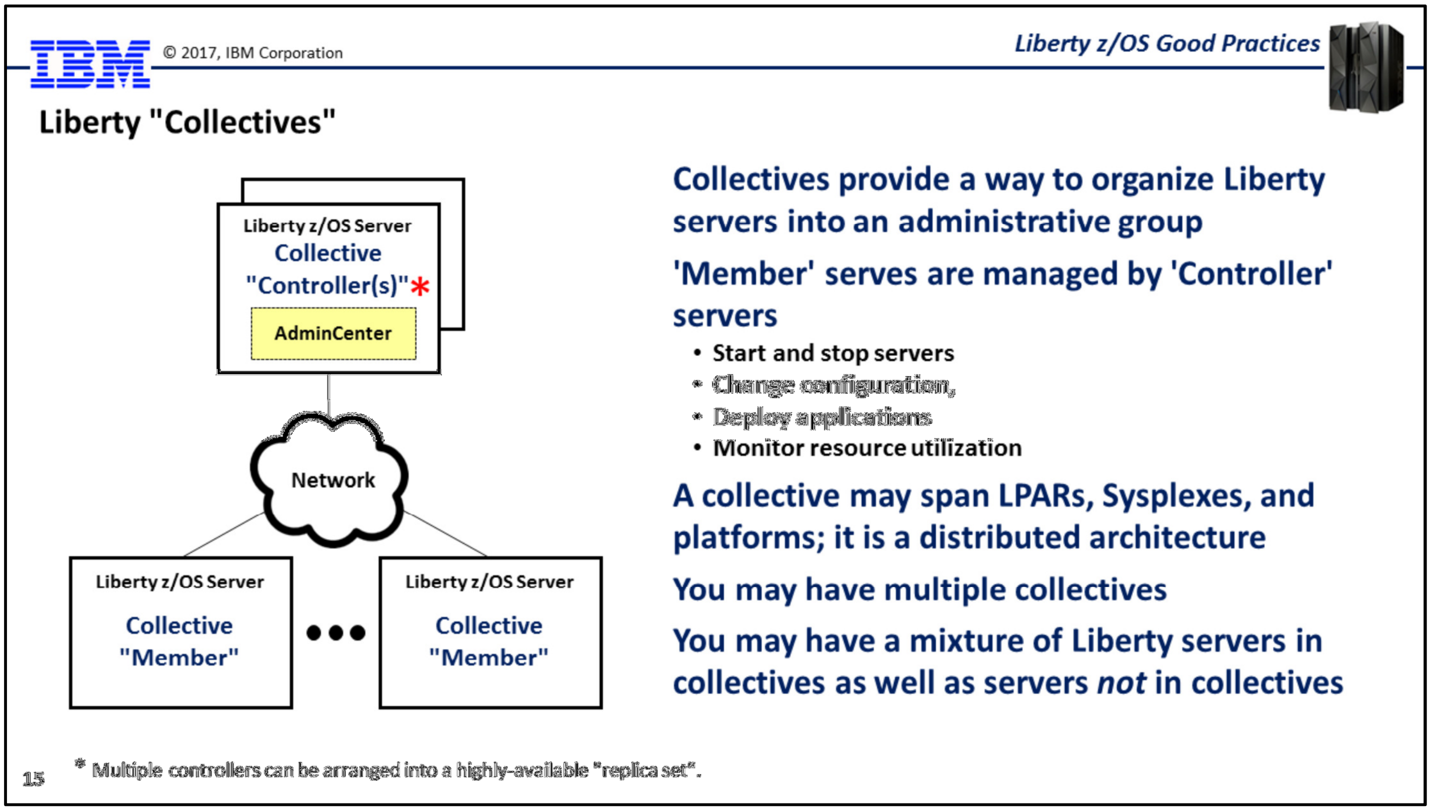
- More than one Angel possible on an LPAR
- This allows operational separation of Angel processes
- Property in bootstrap.properties names Angel server will use

14

Here we introduce a z/OS-specific element of the Liberty story -- the "Angel process." Its role is to provide a means of controlling access to z/OS authorized services by Liberty z/OS servers. The best way to approach this topic is in bullet format:

- The z/OS operating system has certain functions that are deemed "authorized," meaning a server ID must have explicit and deliberately-assigned access to those services. This is to protect other users on the z/OS system that might be affected by improper use of those services.
- The Liberty z/OS Angel Process is a started task that participates in this act of checking for authority to access z/OS authorized services. The Angel Process is a *very* lightweight function: it has no configuration, no Java, no TCP ports, and uses virtually no CPU once started. It's role is to provide the framework for to check whether Liberty z/OS server started task IDs are allowed to access a specified z/OS authorized service.
- The Liberty z/OS Angel Process is required only if you have Liberty z/OS servers on an LPAR that are seeking to use z/OS authorized services. If you have no such Liberty z/OS servers, then you would not need an Angel Process. The z/OS authorized services we are referring to are listed on the chart.

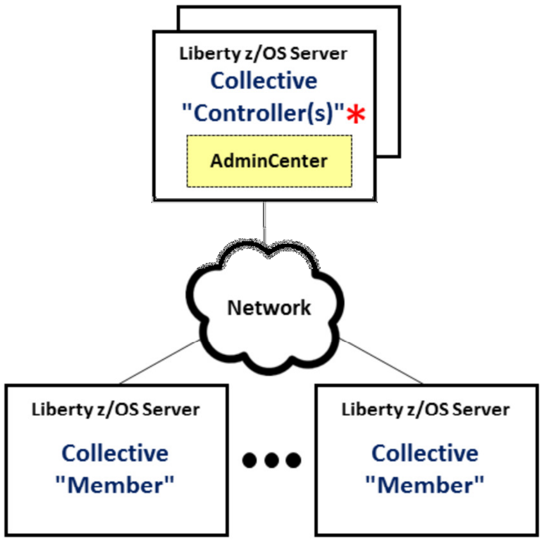
  Note: if you have z/OS 2.1 and you're using the z/OSMF management function, you may already have an Angel process in place. z/OSMF at the 2.1 level and above uses Liberty z/OS as its Java runtime server, and the tasks z/OSMF performs are frequently z/OS authorized services requiring permission to access. So z/OSMF 2.1 involves an Angel Process and SAF profiles so z/OSMF may access those authorized services.
- If an Angel Process is needed, then a *minimum* of one Angel per LPAR is required. However, starting with Liberty z/OS 16.0.0.4 and above, the ability to configure more than one Angel Process was introduced. This was done as a way of creating separation between groups of Liberty servers on an LPAR. Liberty z/OS servers that make use of an Angel for access to authorized services are dependent on that Angel being present. If that Angel is canceled, the dependent Liberty z/OS servers come down. (If the Angel is merely stopped, the stop action will be held up until the individual dependent Liberty z/OS servers are stopped. There is a MODIFY command to list out the Liberty z/OS servers dependent on an Angel process.) The "Named Angel" support in 16.0.0.4 provides the ability to separate this dependency into groups, so one group can take down their Angel without affecting other groups. We mention this here not to make things more complicated, but to alert you to this new function. This modifies the "one Angel per LPAR" rule that's been spoken of since Liberty first came out.
- The name "Angel" is a play on the name for a WAS z/OS traditional task called the "Daemon" server, which sounds phonetically similar to "demon." So the Angel server is the opposite of the "demon" from WAS traditional.



© 2017, IBM Corporation

Liberty z/OS Good Practices

Liberty "Collectives"



Collectives provide a way to organize Liberty servers into an administrative group

'Member' serves are managed by 'Controller' servers

- Start and stop servers
- Change configuration,
- Deploy applications
- Monitor resource utilization

A collective may span LPARs, Sysplexes, and platforms; it is a distributed architecture

You may have multiple collectives

You may have a mixture of Liberty servers in collectives as well as servers *not* in collectives

15

* Multiple controllers can be arranged into a highly-available "replica set".

Liberty on all platforms, including z/OS, has a management construct called "collectives." The concept is that a number of Liberty servers can be arranged and managed through a Liberty server designated as a "controller," with the managed Liberty servers being "members." So we have a "controller" and some number of "members," and together they form a "collective."

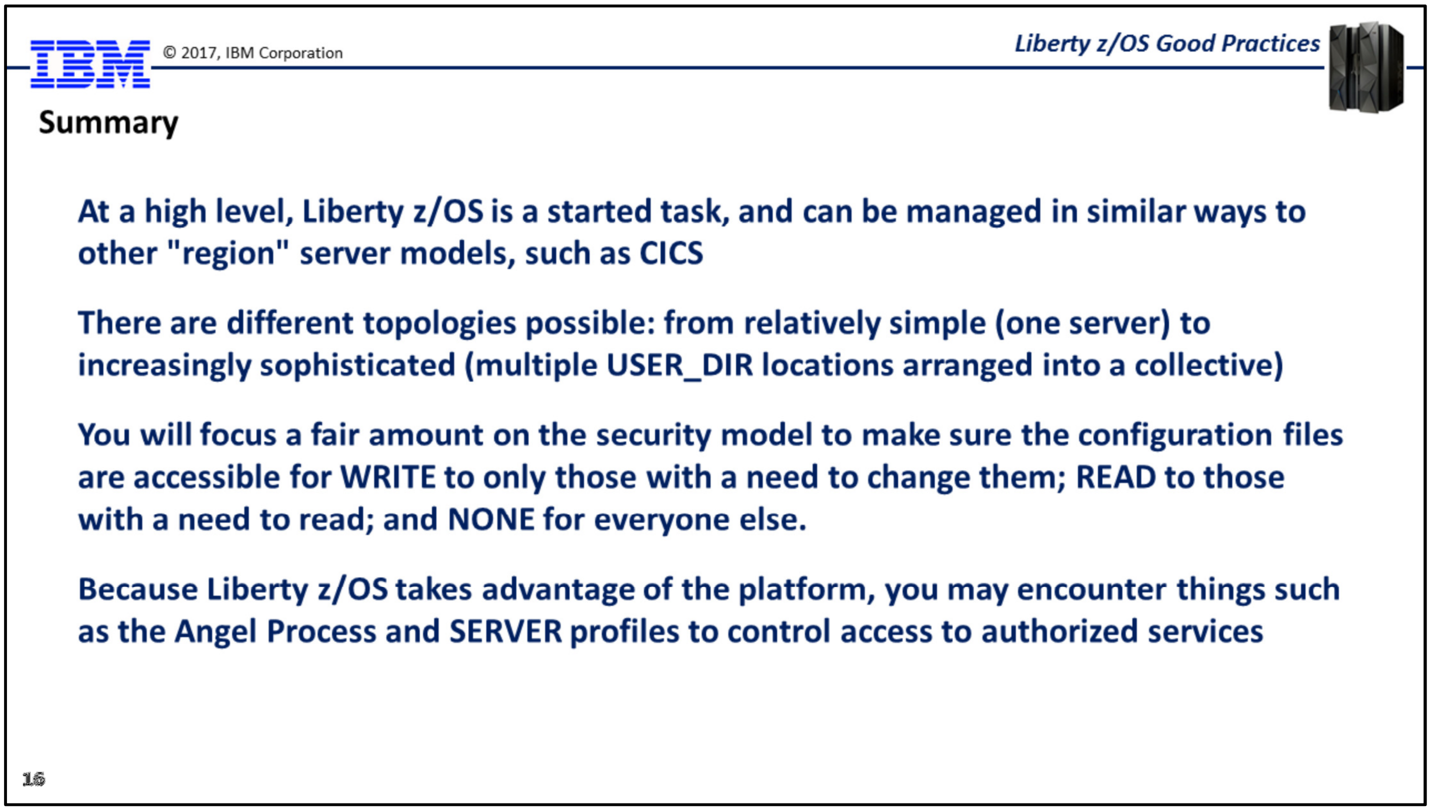
You don't have to use the collective design. You may, if you wish, manage your Liberty z/OS servers individually. The benefit collectives provides is the ability to view those multiple servers through the "controller" for doing things such as starting and stopping servers, changing the configuration, copying application files, and monitoring resource utilization.

A Liberty z/OS server that is joined to a controller to be a member of the collective can be -- quite easily -- removed from the collective. The collective design is quite different from the WAS traditional "node" and "cell" design where membership was somewhat permanent, and remove somewhat difficult.

The collective design is full distributed, which means a collective can span LPARs, Sysplexes, and even platforms. Communications between the controller and its members is over a TCP network.

You may construct between 0 and *n* collectives, based on your needs. And you may have some Liberty servers that are part of a collective and others that are not.

The topic of collectives is fairly broad, so we won't try to get into any details here. We have an entire unit dedicated this topic where the details are explored more fully.



Liberty z/OS Good Practices

Summary

At a high level, Liberty z/OS is a started task, and can be managed in similar ways to other "region" server models, such as CICS

There are different topologies possible: from relatively simple (one server) to increasingly sophisticated (multiple USER_DIR locations arranged into a collective)

You will focus a fair amount on the security model to make sure the configuration files are accessible for WRITE to only those with a need to change them; READ to those with a need to read; and NONE for everyone else.

Because Liberty z/OS takes advantage of the platform, you may encounter things such as the Angel Process and SERVER profiles to control access to authorized services

16

We've reached the summary chart, which means we're done with this unit.

This unit was designed to convey some of the "key" concepts of Liberty and Liberty z/OS. The intent was to provide enough of a foundational understanding that the details supplied in the other units have some context.

The nature of Liberty provides a fair degree of flexibility to design your use of it in many different ways. There is no "one" way to use Liberty.

Finally, on z/OS Liberty takes on some z/OS things -- the started task, SAF profiles, and the Angel Process. Liberty z/OS is still very much like Liberty on other platforms, it's just there's an ability to take advantage of the z/OS platform and with that comes some z/OS elements to the design of the Liberty runtime environment.

End of Unit