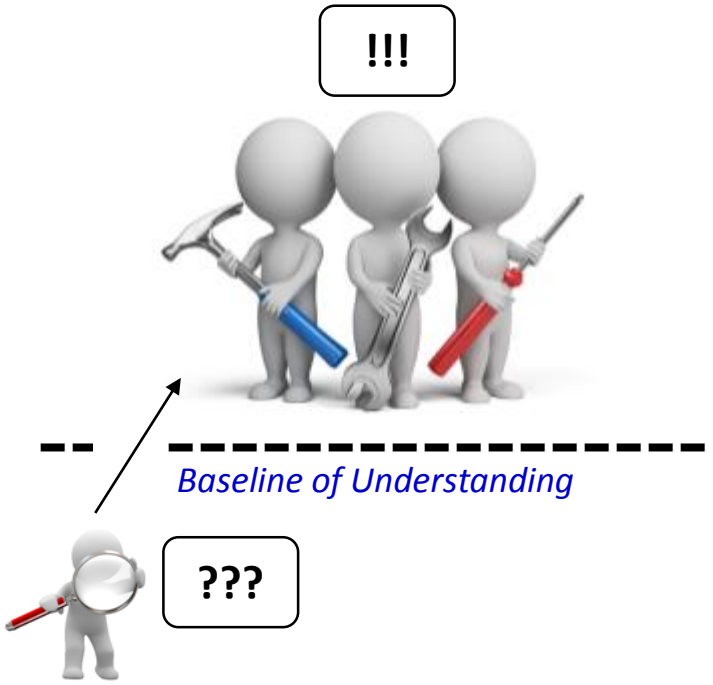
WebSphere Liberty z/OS

A review of key concepts

Objective of this Presentation



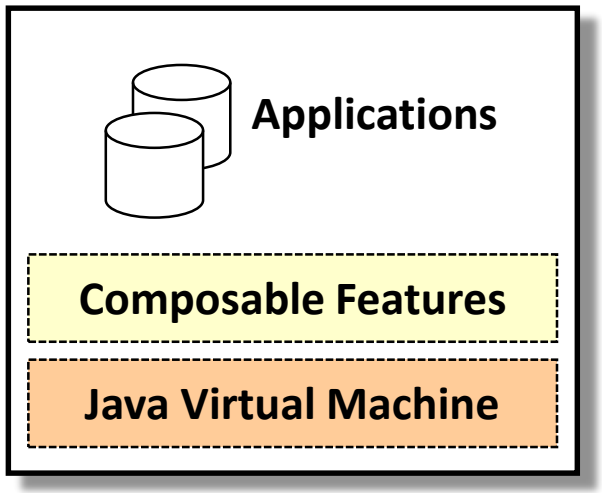
Provide a set of key concepts and principles of Liberty z/OS that will help with the details that will follow

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

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

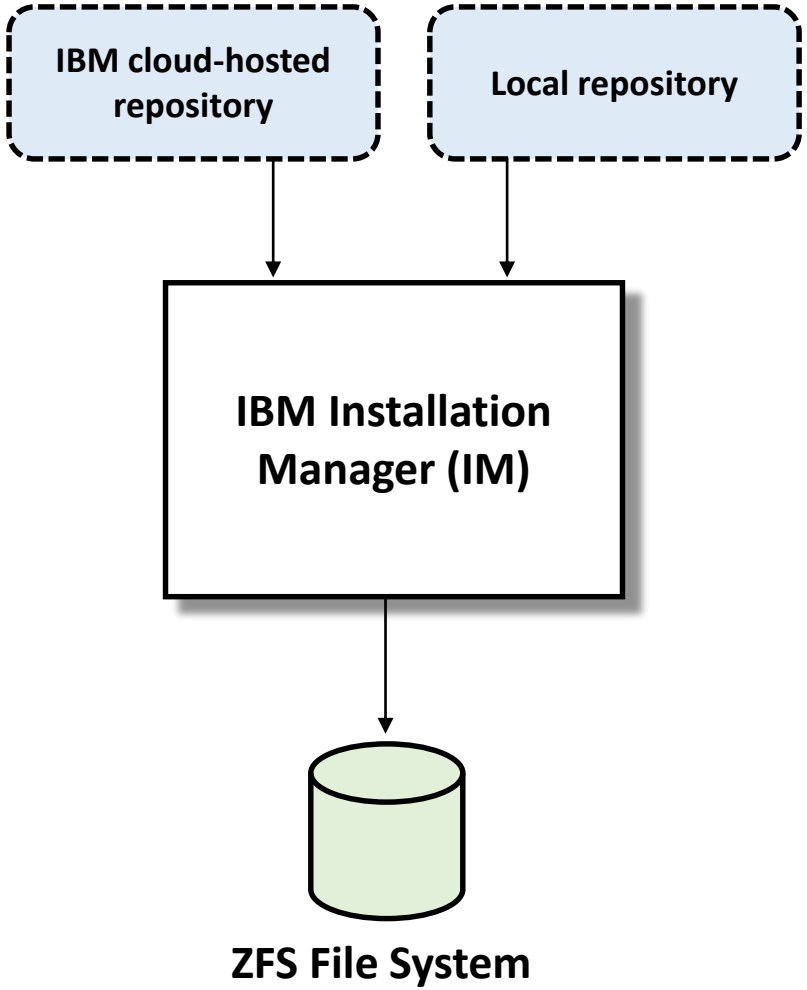
On z/OS it can be run as a Started Task

- 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

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

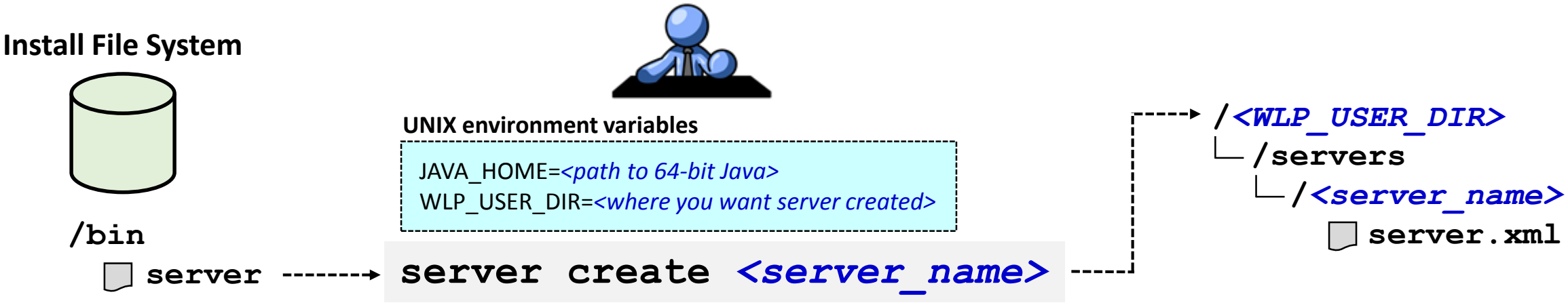
Result is a ZFS File System with Directories and Files

- 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

Creating a Server



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

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

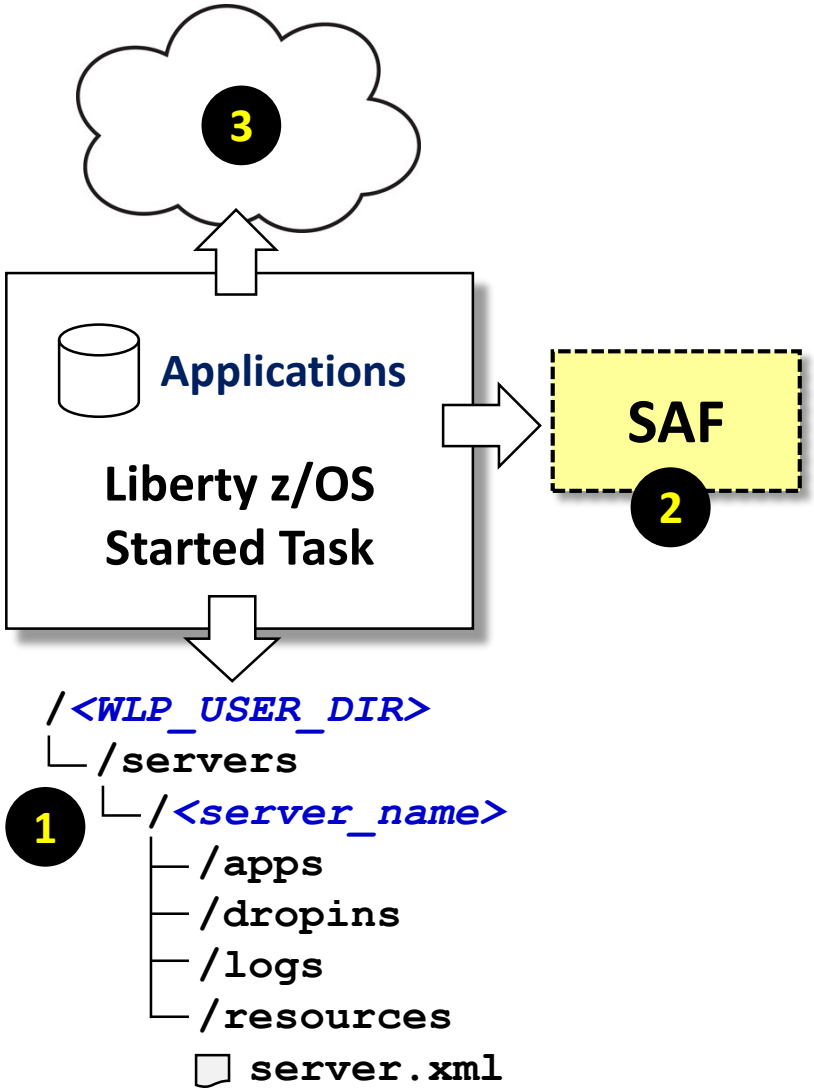
Other configuration files that may be used

- 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

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

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>*

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

"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

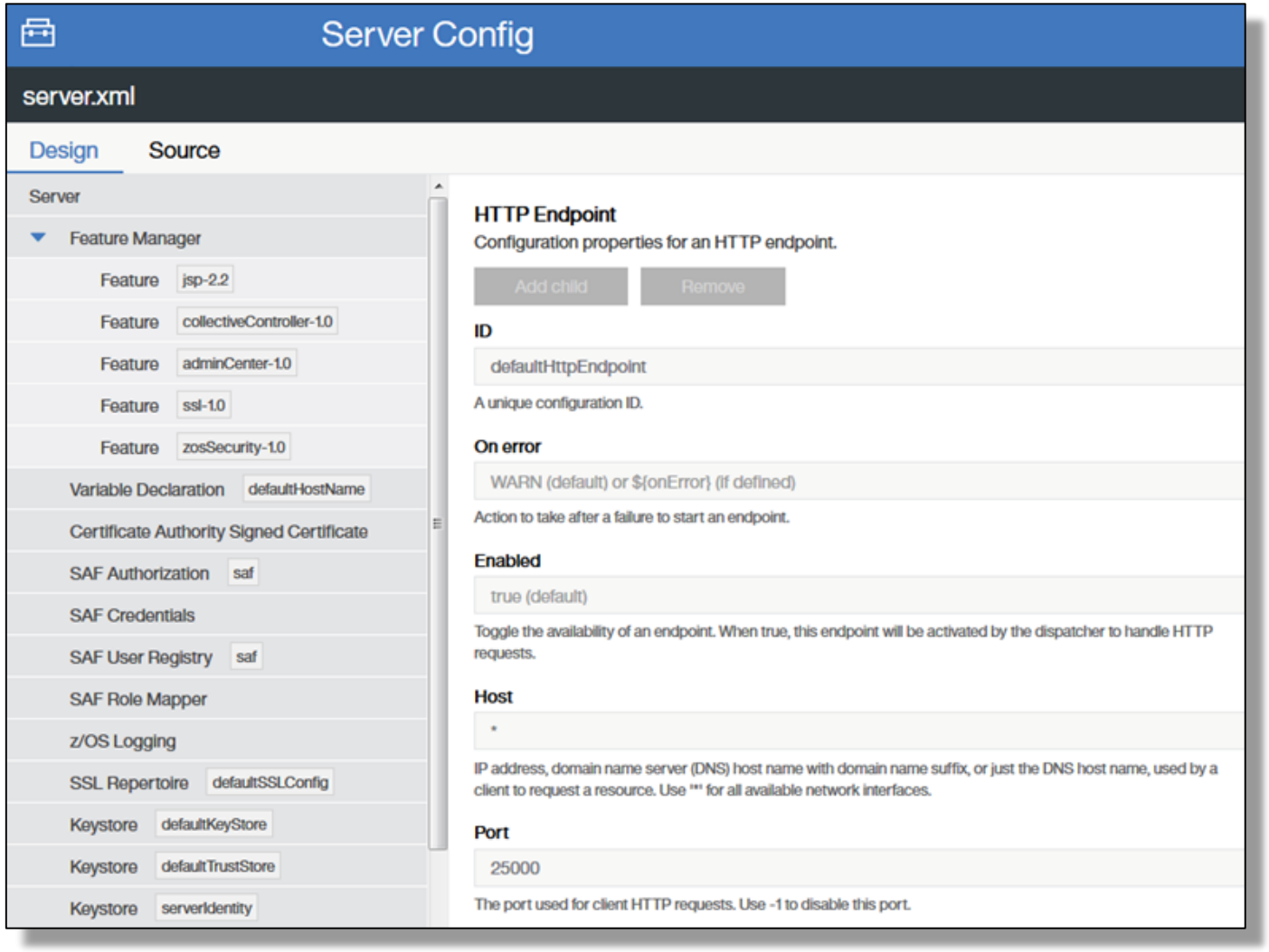
```
<server>
 (XML configuration elements to be included)
</server>
```

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

AdminCenter Server Configuration Wizard



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

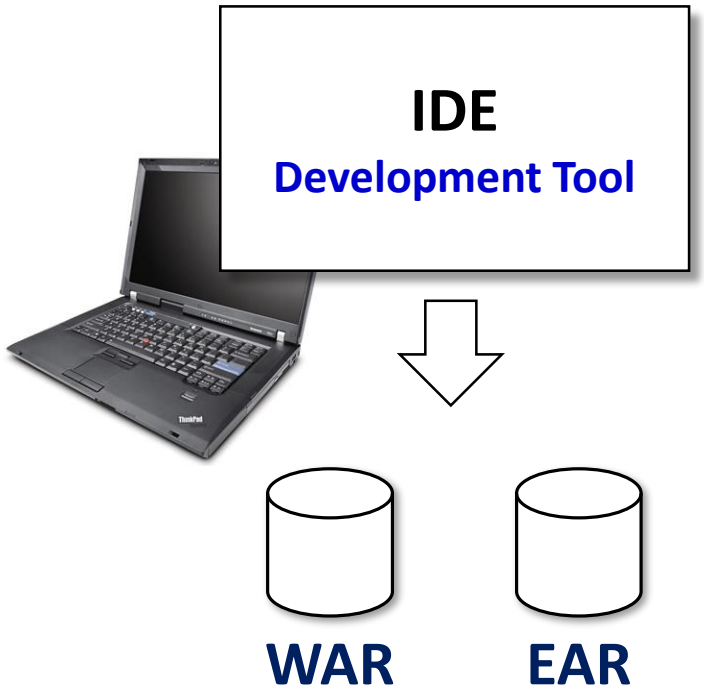
It provides a browser-based graphical interface

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

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

Application Deployment

Liberty z/OS Server

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

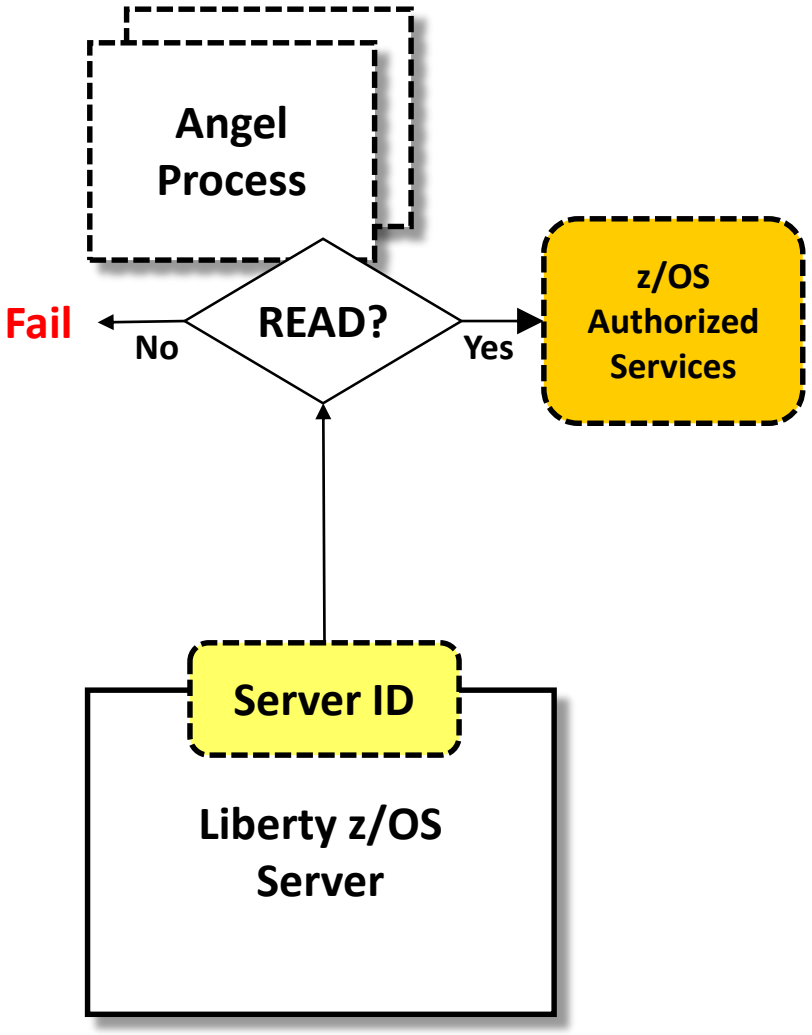
Two essential ways to "deploy" an application:

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

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

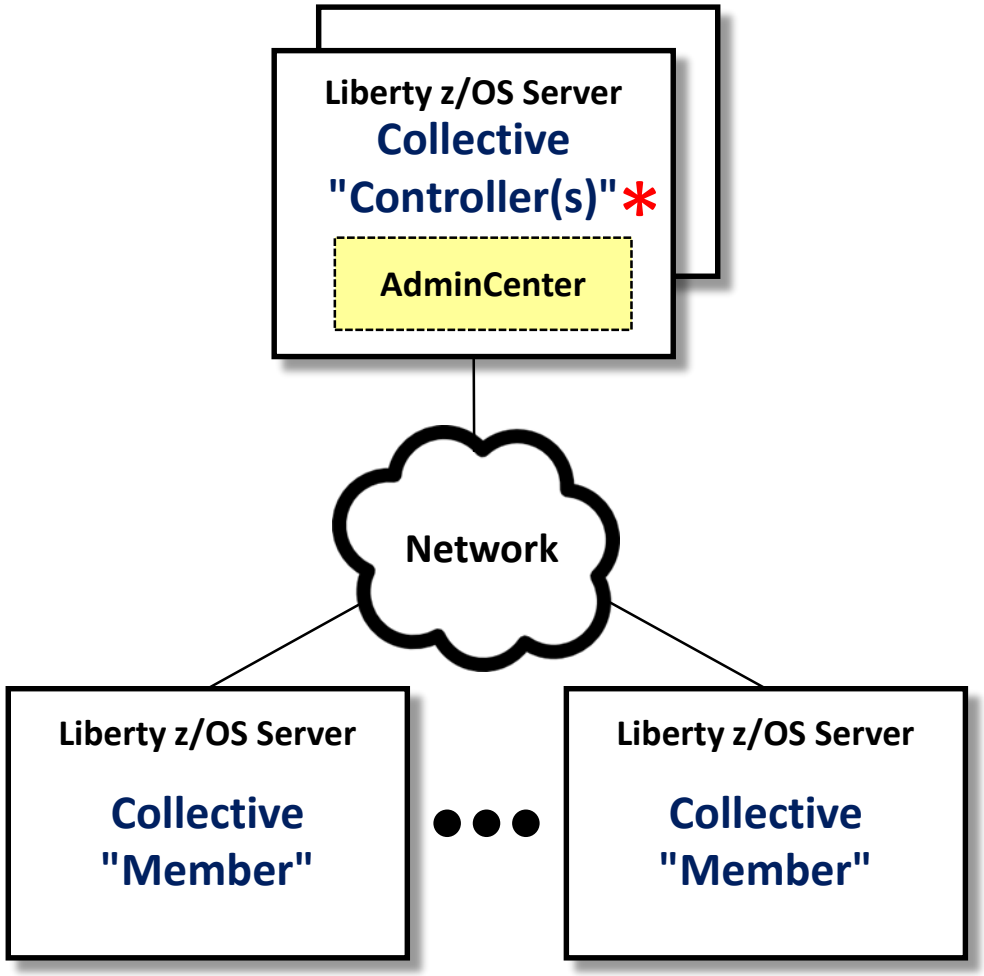
Features that require the Angel Process

- 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

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

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

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