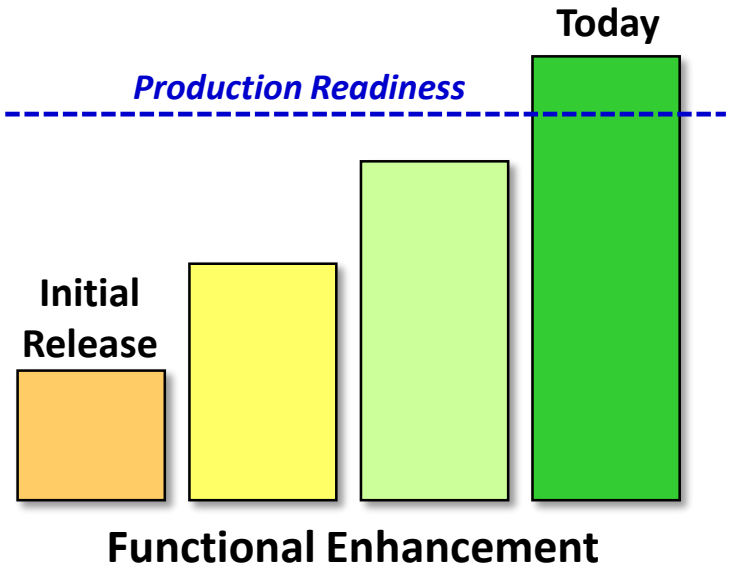
WebSphere Liberty z/OS

An Overview of Good Practices

Liberty is Production Ready



Liberty was first introduced four years ago

Initially it was lacking many features; okay for development and some testing, but not production

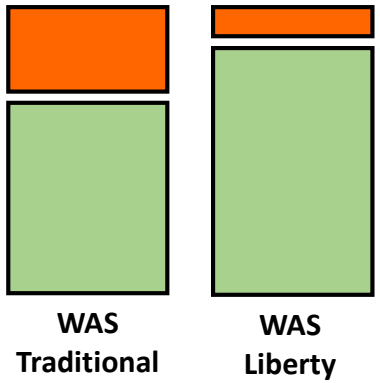
Liberty operates under a "continuous delivery" model

Over time it has received many functional updates

Today it is a full Java EE runtime server

We are seeing increasing interest in -- and in some cases adoption of -- Liberty for production workloads. Its time has arrived.

Liberty Offload and Performance



zIIP Offload

- **WAS Traditional saw an approximate (and average) 80% zIIP offload rate**
- **Liberty sees more zIIP offload**
- **Liberty's single JVM model implies less native code, and therefore less GP**
- **WAS Traditional server (CR+SR) ~ 1GB memory; Liberty is composable = less**

Liberty z/OS Performance*

35%

DayTrader 3 benchmark EJB application saw a 35% performance improvement. This reflect general performance improvements in Liberty.

30%–40%

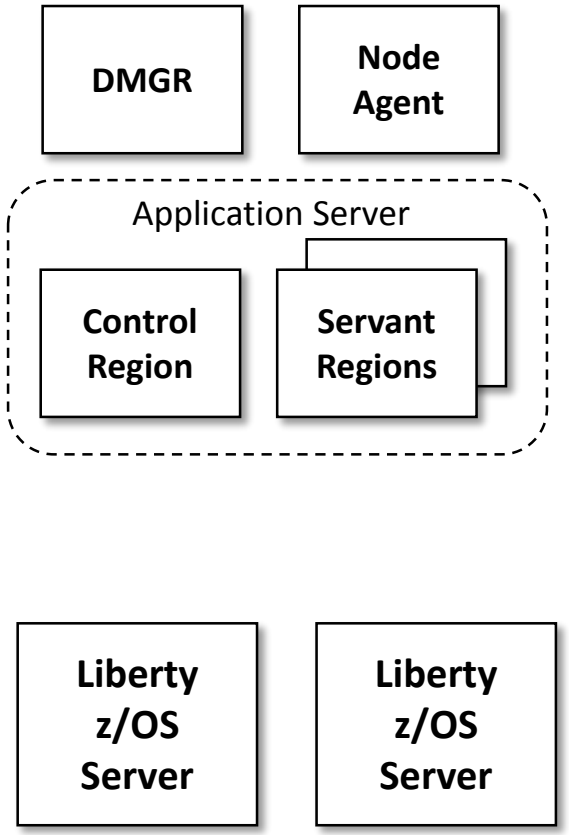
Improvement using Liberty z/OS Asynchronous I/O vs. Network I/O for 2000, 4000, 8000 concurrent users. Reflects leveraging platform for scalability.

4X

Batch program using inbound WOLA against simple ping application. Reflects streamlined code path in Liberty vs. CR/SR message handling in WAS Traditional.

* Performance results are based on controlled tests in a specific environment. Your results may vary. These results do not represent a promise of similar results.

Liberty z/OS and WAS Traditional z/OS



WAS Traditional has been an effective production work application server for many years

Liberty can coexist with WAS Traditional

- Just be aware of things such as port conflicts, and naming that may imply conflicts with things like JCL procs, or SAF profiles

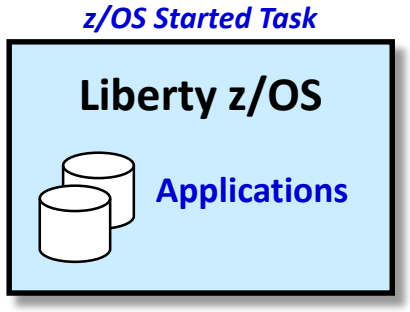
Work presently running in WAS Traditional may remain if there is a compelling business reason to stay

- Applications make use of APIs in WAS Traditional not found in Liberty
  More on this later when we discuss application development
- Other reasons to remain on WAS Traditional, such as automation built around WSADMIN scripting

Consider Liberty for new workloads, or for workloads that do not require the functionality of WAS Traditional

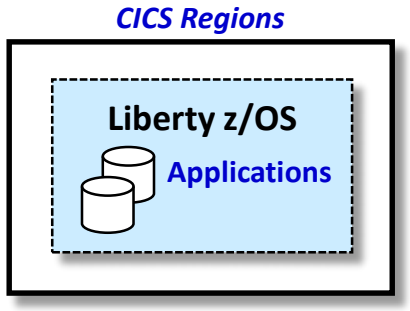
This is not an either/or consideration ... it can be both

Different Ways Liberty z/OS Can Be Operated



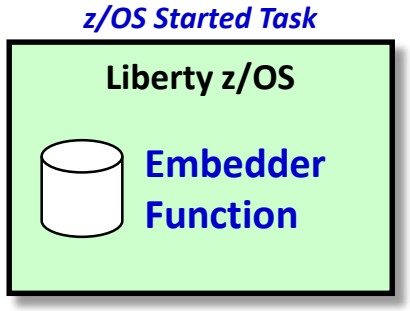
"Application Server"

- Liberty z/OS is entitled through WAS z/OS license
- Your applications are deployed and run inside this server runtime environment
- *This is the focus of this material*



"Inside CICS"

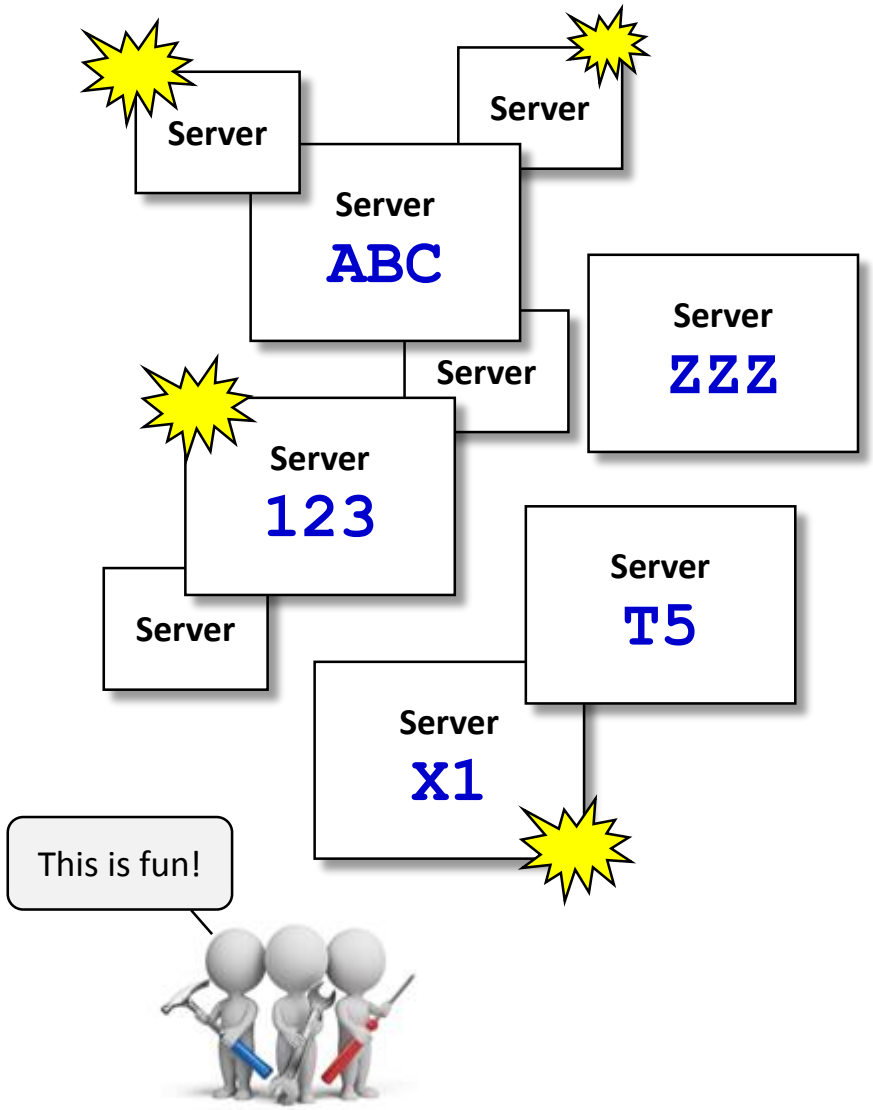
- Liberty z/OS operates inside a CICS region; entitled with CICS license
- The programming interfaces are the same
- The operational model is aligned with CICS operational practices



"Embedded"

- Liberty z/OS is used as a Java runtime for some IBM or vendor function
- IBM or the vendor supplies Liberty; license entitlement under their license
- Liberty z/OS server is typically dedicated to that function
- Examples: z/OSMF 2.1, z/OS Connect EE V2.0

Objective #1: Avoid Confusion from Unplanned Server Proliferation



Without some care, it is possible to have a number of Liberty servers come into existence

- It's very easy to create a new server if you know how

Those servers may have locations, names, and ports that are all different

- There is considerable flexibility in these things

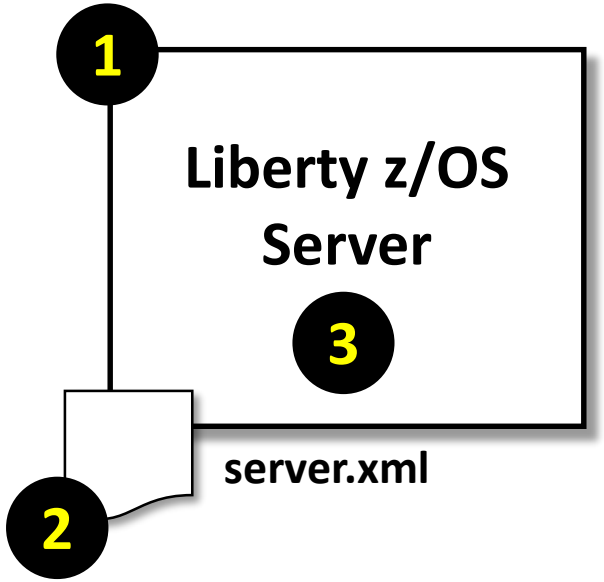
That may be okay for ad hoc development and test, but would surely be a problem for production

- Production environments is what we're focusing on here

If you wait until after-the-fact, you'll be left trying to take inventory of everything that was built

Better to approach Liberty z/OS with some planning up front, rather than after-the-fact

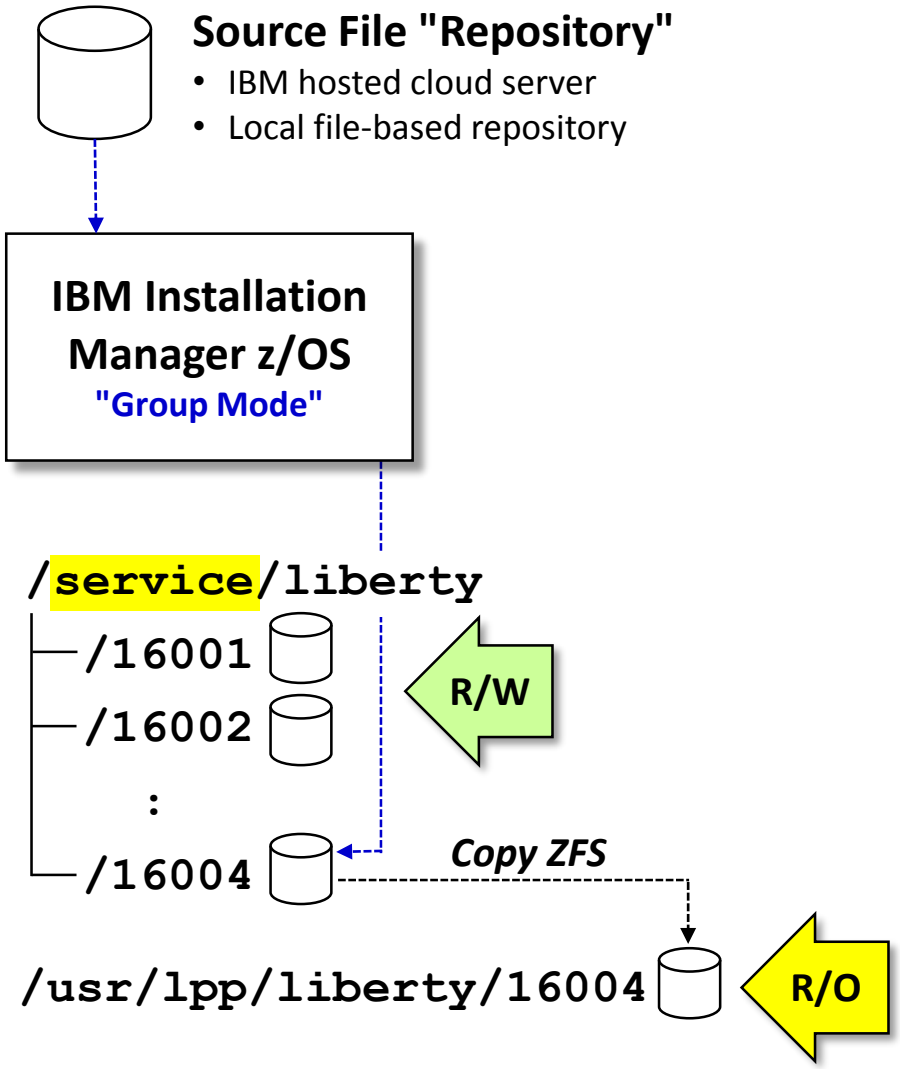
Objective #2: Protect Against Unauthorized Activity



1. **Prevent unauthorized server creation**
   - This is based on UNIX permissions in install /bin directory
   - Then authorized people create servers according to the plan
2. **Prevent unauthorized modification of configuration**
   - This is accomplished by carefully planning out file system ownership
   - Then UNIX permissions for 'group' and 'other'
3. **Prevent application access to configuration**
   - This is accomplished by separating the task ID from the configuration file system owner ID, and managing UNIX permissions

That's the essential file protection security; after that comes the standard application layer security: encryption, authentication, authorization

Liberty z/OS Installation



Use "Group Mode" for IM on z/OS

- Makes best sense on a multi-user system such as z/OS

Employ a "service zone" concept for installations

- Allows IM R/W access to the file systems it writes to
- iFixes can be installed without concern about running servers

Maintain installs by versions of Liberty

- Installs are faster
- Provides for easier fall back if new level proves unsatisfactory

Copy out and mount R/O for server access

- Prevents any tampering with the files

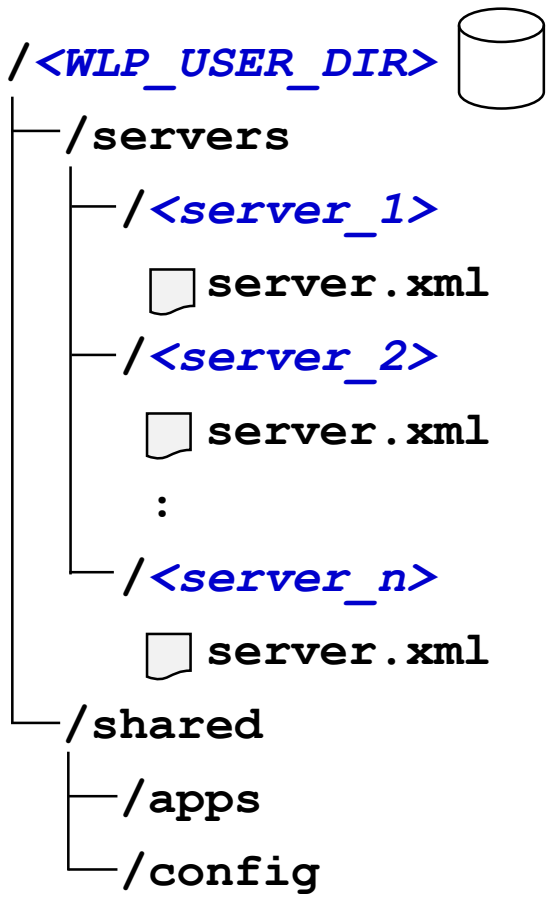
Make sure 'other' permission bits are 0

- Only group members may read files and operate servers

Server STC IDs are then connected to the group

- This allows the server ID read, which is all it needs to load the files when a server is started

The Liberty WLP_USER_DIR



Server are created under the WLP_USER_DIR you name

- UNIX environment variable; 'server create' command creates servers there

Mount a file system at the WLP_USER_DIR

- Do this before creating servers; provides an easy backup/restore point
  Size needed depends on many factors (application sizes, log sizes, etc.)

You may create any number of servers under a WLP_USER_DIR

- Servers under a WLP_USER_DIR have access to the common /shared directory
- It's possible to use the same JCL start proc for multiple servers

Servers under a WLP_USER_DIR should be related in some way

- Not a technical requirement; rather a good practice

Servers under a WLP_USER_DIR should have consistent names

- This gets to the naming convention, which we discuss on the next page

Servers under a WLP_USER_DIR should operate on same LPAR

- Not a strict requirement; this simply avoids cross-LPAR write operations

You may create multiple different WLP_USER_DIR locations

- This provides a way to segregate different groups of servers: DEV, TEST, QA, etc.

Naming Convention Overview

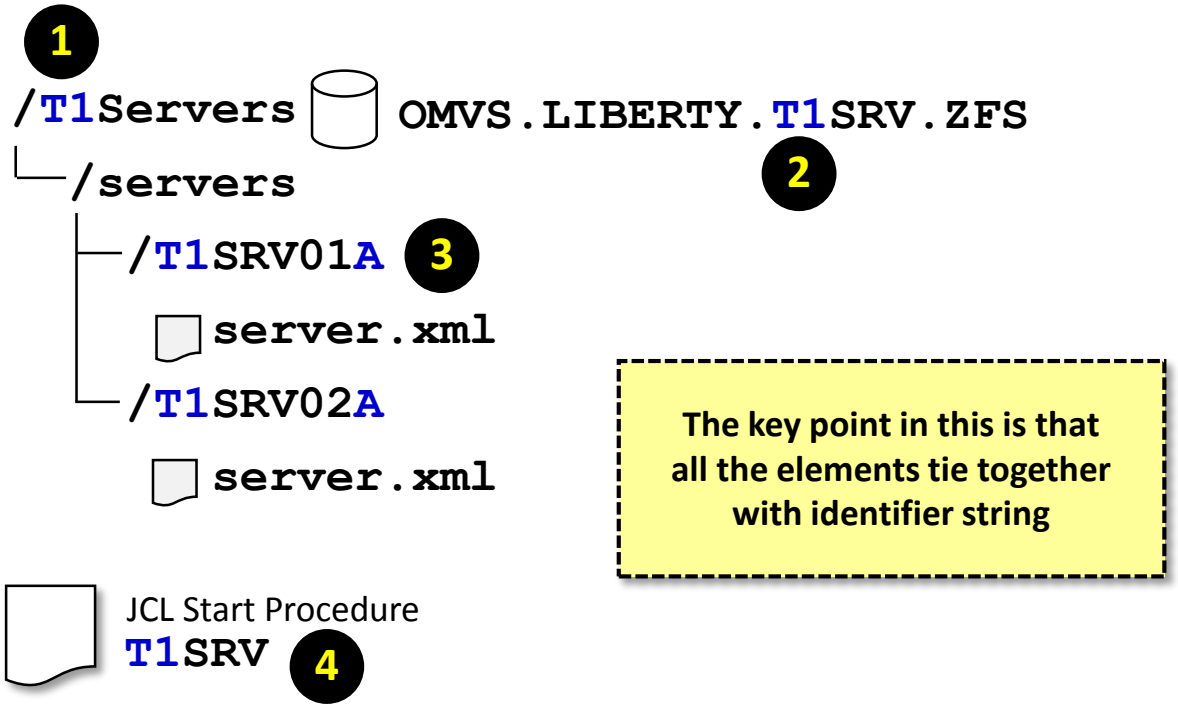
There is no "one" naming standard that is "best" ... there are guidelines, but ultimately it's what best serves your needs.

When you are coming up with your naming convention, it is a good thing to "paper test" it to see if it holds up to different anticipated scenarios.

A good naming standard achieves a few things:

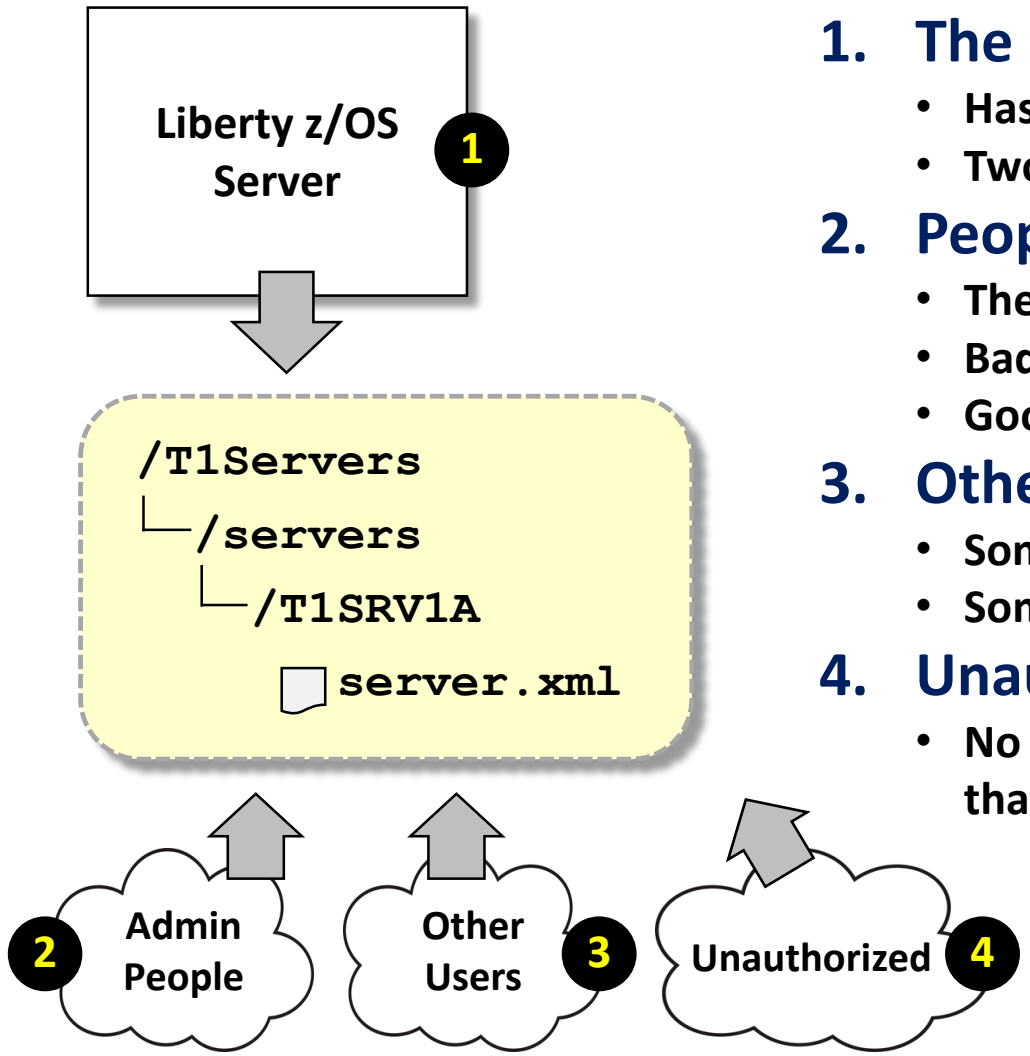
- **It allows you to quickly understand what a server's purpose is, based on the name**
- **It allows you to quickly determine what servers are related to one another, based on the name**
- **It relates different naming artifacts based on the convention used**
  For example: the Server Name relates to the JCL start procedure name, which relates to the STC JOBNAME, which is tied to SAF artifacts such as the task ID

Naming Convention -- *an Illustration*



1. **WLP_USER_DIR**
   - Start directory name with two (or three) character identifier string that is related to purpose of servers
2. **User Directory File System**
   - Mount a ZFS at this location and include identifier
3. **Server Names**
   - Server names starts with the same identifier string
   - Limit to 8 characters to they can align with z/OS limits
   - Uppercase to better match START command
4. **JCL Start Procedure**
   - JCL start proc starts with same identifier string
   - May use generic JCL for all servers under user directory
   - If each server has its own JCL, then make equal to server name
5. **ID/Group Assigned to STC**
   - The ID and group assigned starts with identifier string
6. **START command, JOBNAME value**
   - If generic JCL used, then add JOBNAME equal to the server name
   - If server-specific JCL used, then add PARMS= to the JCL and drop from the START command

Those Seeking Access to the Configuration File Structure



1. The Liberty z/OS Server

- Has a need to both READ and WRITE
- Two options: server ID owns files, or is a separate ID from file owner

2. People Responsible for Administering the Server(s)

- They have a need to both READ and WRITE
- Bad practice: sharing login ID/password
- Good practice: using SAF SURROGAT to switch to file owning ID

3. Other People Related to the Server's Activities

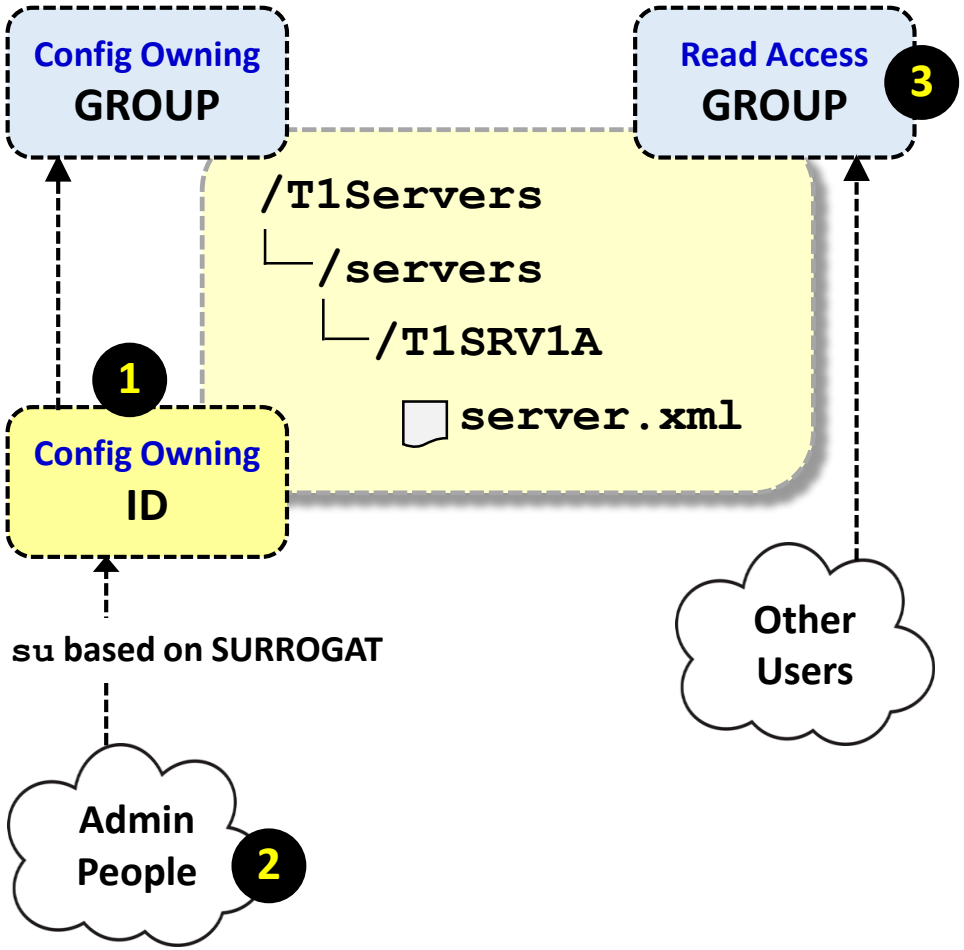
- Some have READ only ... logs, etc.
- Some may need limited WRITE ... then use "include" processing

4. Unauthorized People

- No access at all; key is insuring these fall into 'other' permission bit, and that is marked as '0'.

At the heart of this is a discussion of the UNIX file *owner*, *group*, and *other* permissions

Guiding Principle: *WRITE* through Owner; *READ* through a Separate Group



1. The Configuration Owning ID

- This ID is what's used to create the server

2. Administrator WRITE Access

- They should not share the owning ID password
- They can gain write access via switching to the ID
- The SAF SURROGAT profile can be used to control this

3. Separate Configuration Group

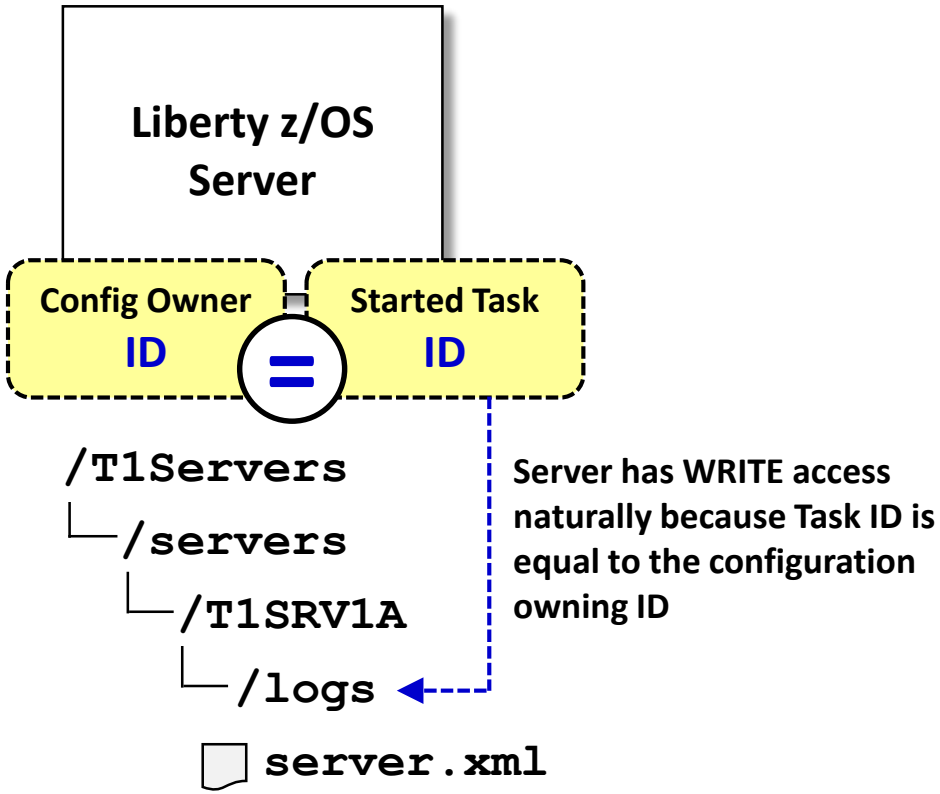
- Separate group is created and made the configuration group via a `chgrp` command

  You could simply connect READ users to the owning group, but if that group is ever granted other SAF access, then those READ users would inherit that access. Safer to have a separate group.
- Users with a need to READ are connected to this group

Creating Servers: Is Task ID the Same as the Configuration Owning ID?

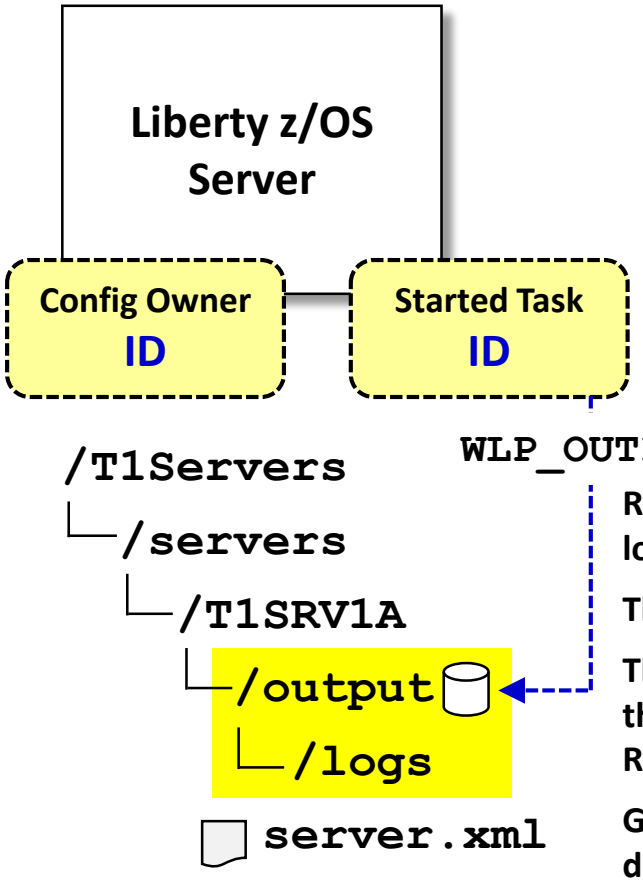
They *can* be the same; but an argument can be made they should not be the same -- different ID prevents application from changing configuration

Task ID Same as Config Owner ID



Server has WRITE access naturally because Task ID is equal to the configuration owning ID

Task ID Different from Config Owner ID



Note: the details* of this are spelled out in the 'Security' unit.

`WLP_OUTPUT_DIR`

Re-directs server to write its output to the location specified by the variable.

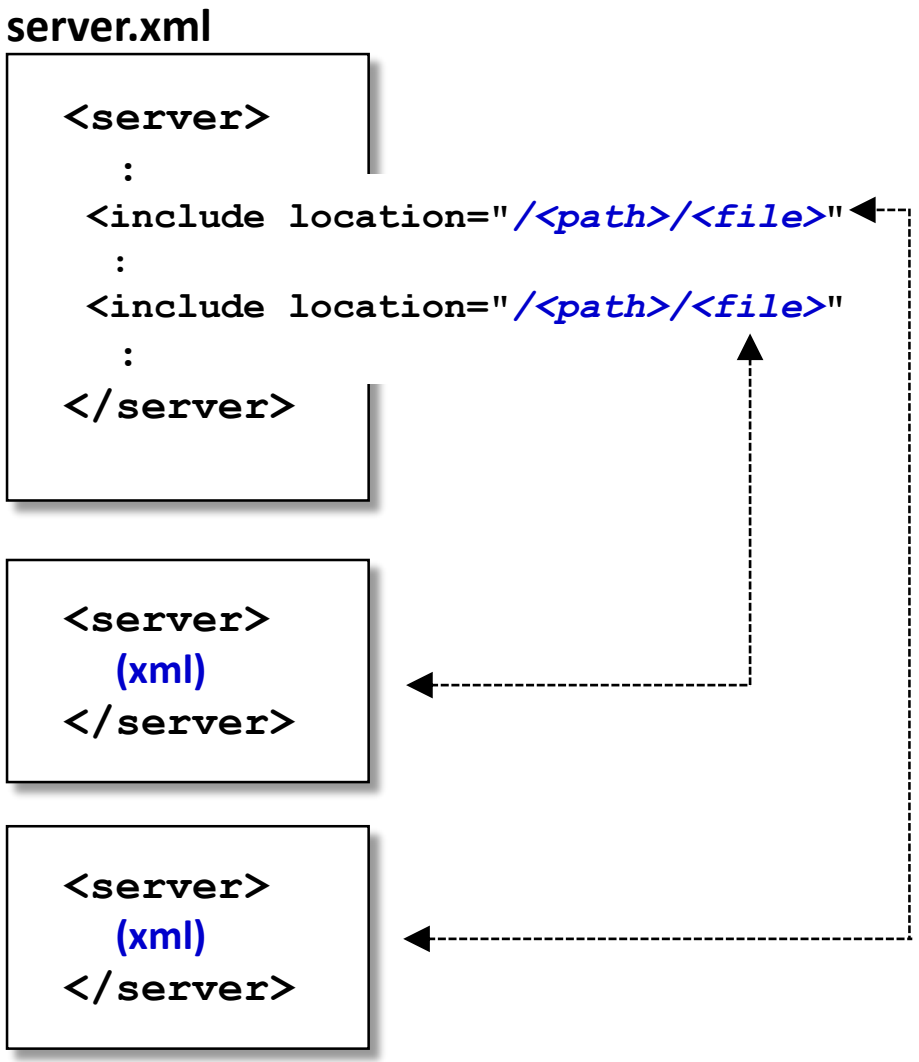
That location can be owned by the task ID.

The rest of the configuration is owned by the Config Owner ID, and the Task ID has READ through group access.

Good practice: mount ZFS at output directory to contain large log output or verboseGC output

* Notably, you need to specify the WLP_OUTPUT_DIR for the server *before* the server is started for the first time. That way the /logs, /resources, and other output directories are created under the task ID at the output directory location

Include Processing



Provides a way to externalize configuration elements from the main `server.xml`

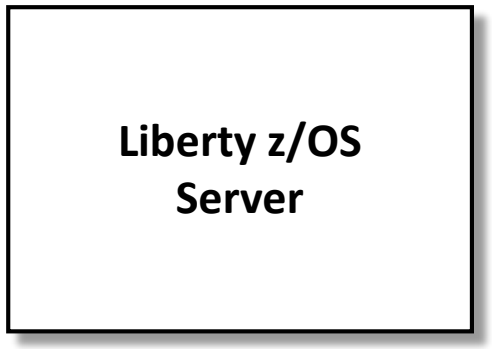
Minimum requirement: STC ID needs READ access

Potential "good practice" uses for "include" files:

- Maintain any common XML in a central location and include in any servers that need it
- Maintain any configuration XML related to security (i.e. LDAP information, or XML with encoded passwords) in separate files that are tightly controlled
- In restricted test environments, provide testers write access to include files but not main server.xml; this allows them to make their updates without granting access to the whole
  There exists "onConflict" rules that can prohibit the overwrite of configuration elements by elements coming in on an include statement

It's a useful facility ... but excessive use, including too much nesting, can create an environment where understanding the configuration is challenging. Use, but use wisely.

TCP Port Management



A Liberty server can have between 0 and *n* TCP ports defined. It depends on what the server is configured to do.

Because Liberty is so flexible in how it can be configured*, it will be challenging to determine with precision how many ports will be needed

At a minimum you should set aside a range of TCP ports for the servers under a given WLP_USER_DIR, then allocate from that range for the servers.

For production servers in particular, consider using SAF SERVAUTH to protect use of the defined TCP ports for that server.

Keep track of TCP ports set aside and used using whatever tool best suits

* WAS Traditional servers used a lot of TCP ports, but the number was known ahead of time. So planning for the TCP port usage there was actually easier.

Controlling Dynamic Updates

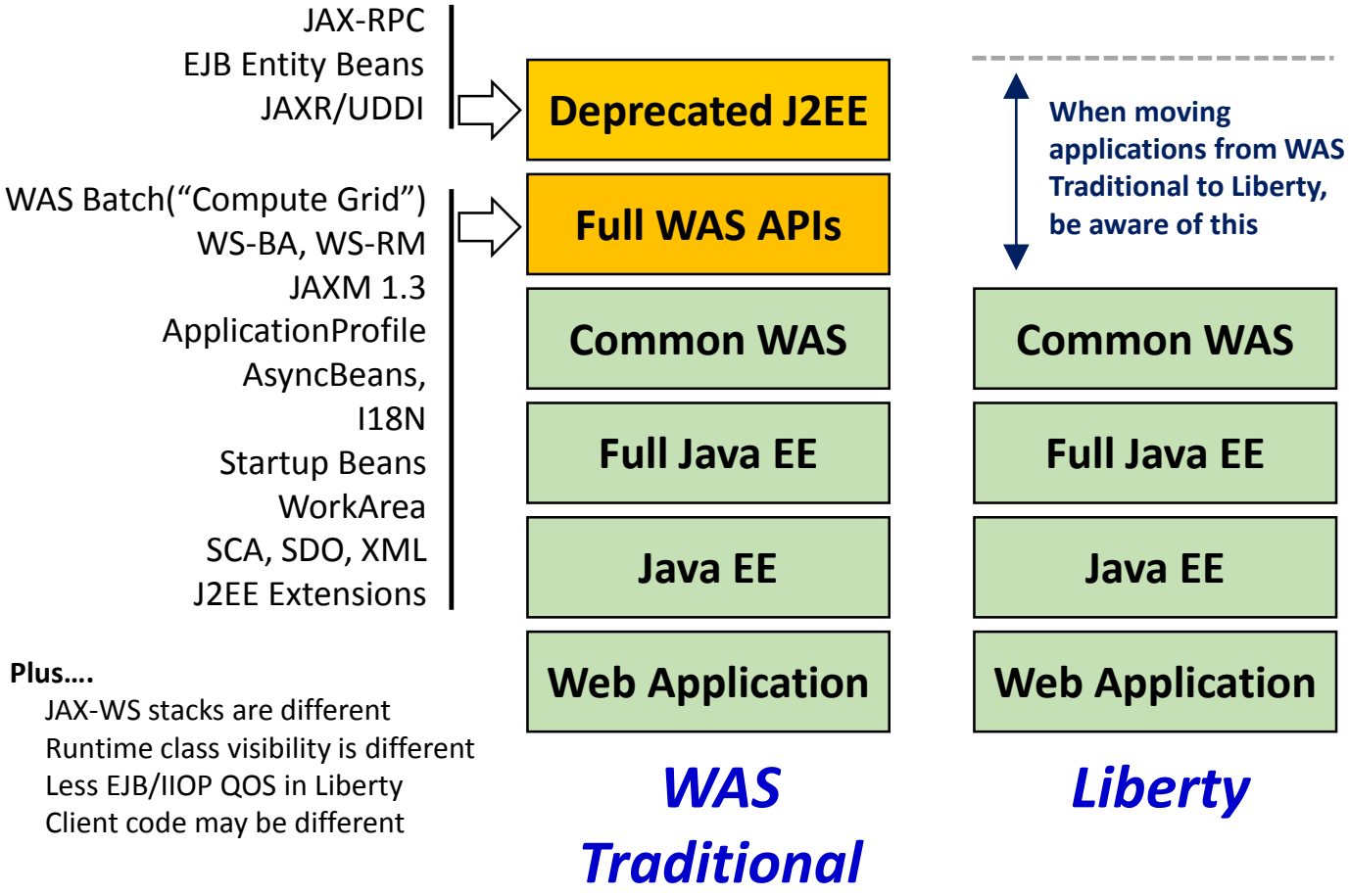
By default, Liberty maintains polling threads that watch for changes to configuration and application, and dynamically updates when changes are detected.

The default polling interval is 0.5 seconds

In a production environment you will likely not want dynamic updates on. Better to rely on scheduled server restarts. This is particularly true when "rolling" updates across a clustered environment.

```
<!- Disable dynamic updates and reduce CPU -->
 <applicationMonitor dropinsEnabled="false" updateTrigger="disabled"/>
 <config updateTrigger="disabled"/>
 <automaticLibraries monitorEnabled="false"/>
 <cdi12 enableImplicitBeanArchives="false"/>
```

Application Development and Migrating Application from WAS Traditional



The application programming APIs for Liberty are the same across all platforms

When considering moving application from WAS Traditional to Liberty, be aware of deprecated APIs and other APIs not present in Liberty

Normal Java coding best practices apply:

- **z/OS is typically a higher-volume processing platform, so design/write code accordingly**
- **Heavily used methods should be inspected carefully to make sure they are as efficient as possible**
- **Profile applications before deploying into high-volume z/OS environments**

Application Deployment

Liberty z/OS Server

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

Deploy Application File Using Any Deploy Mechanism You Wish

- Ultimately this is simply uploading an application package file (EAR or WAR) to a file system location and letting the server know about the location

Deploy with No Explicit Definition

- You could simply drop application file into **/dropins** directory, or another directory you designate as the dropins location (`<applicationMonitor dropins=""/>`) element
- In a production environment, you should disable dynamic updates (see previous chart)

Deploy with Static Definition

- You can deploy the application into any directory and point to it from server.xml:

```
<application location="<app_name>" />
```
Either the /apps directory or the /shared/apps directory

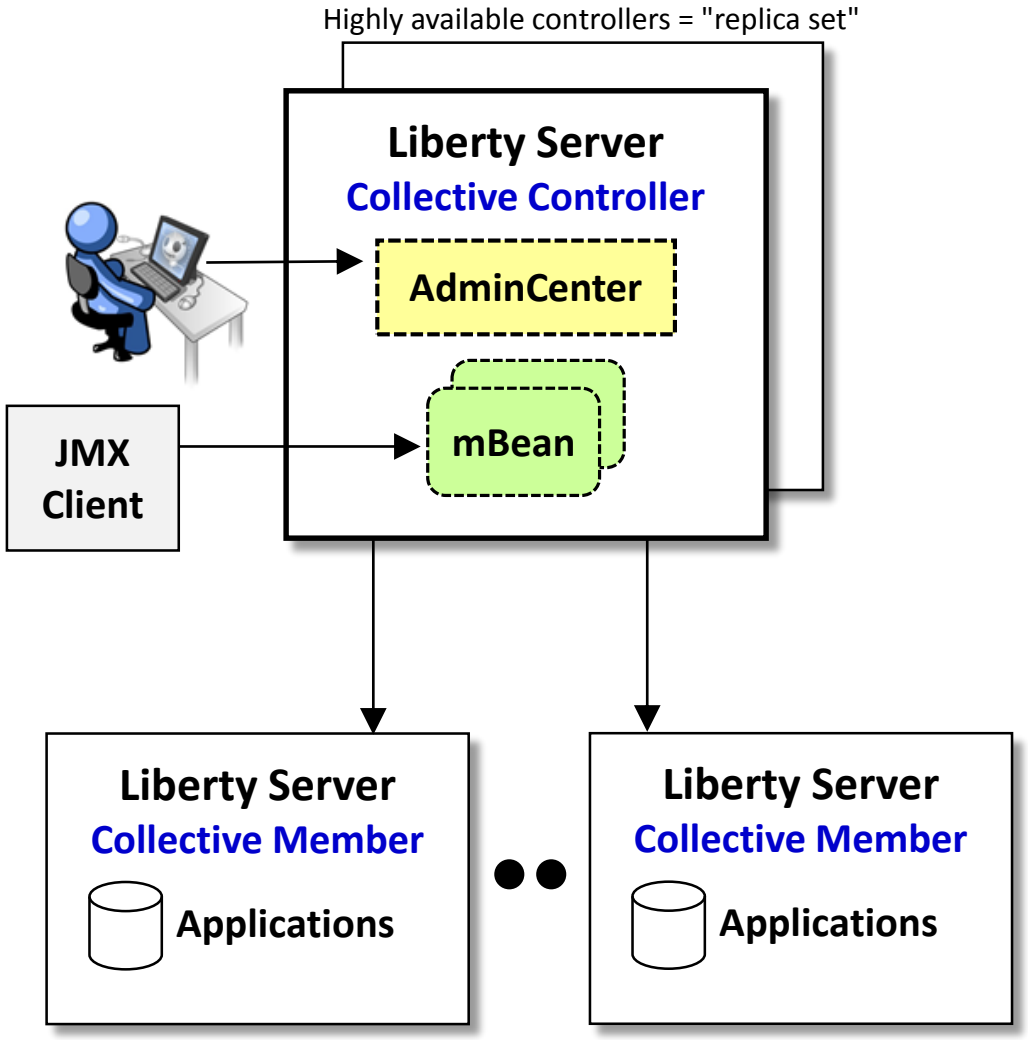
```
<application location="${server.config.directory}/apps/<app_name>" />
```
The /apps directory

```
<application location="${shared.app.directory}/<app_name>" />
```
The /shared/apps directory

```
<application location="/<full_path>/<app_name>" />
```
Any path location the server ID has READ access to

Static definition with deliberate server restart is better in a production setting

Collectives



Liberty Collectives ...

- Provide a way to organize Liberty servers into a logical grouping for administrative purposes.
- Collectives are somewhat flexible in that servers can be added or removed rather easily.
- Primary benefits of collectives:
  - You can manage servers from a centralized interface. "Manage" implies starting/stopping, making configuration changes, pushing files out to servers, etc.
  - You can take advantage of intelligent routing and auto-scaling technologies
- Collectives are built on:
  - Optional use of SSH for cross-server command invocation. (It is possible to use collectives if SSH is not present.)
  - Mutual authentication using SSL certificates, which may be held in file-based key/trust stores or in SAF keyrings

Liberty Collectives Good Practices

Understand that collectives are not required; use them where they provide benefit

- Provides a centralized point of control for multiple servers (start/stop, monitor, etc.)
- Provides a mechanism for generation of HTTP server plugin-cfg.xml
- Provides a mechanism for intelligent routing to clustered Liberty servers, and dynamic scaling of Liberty servers

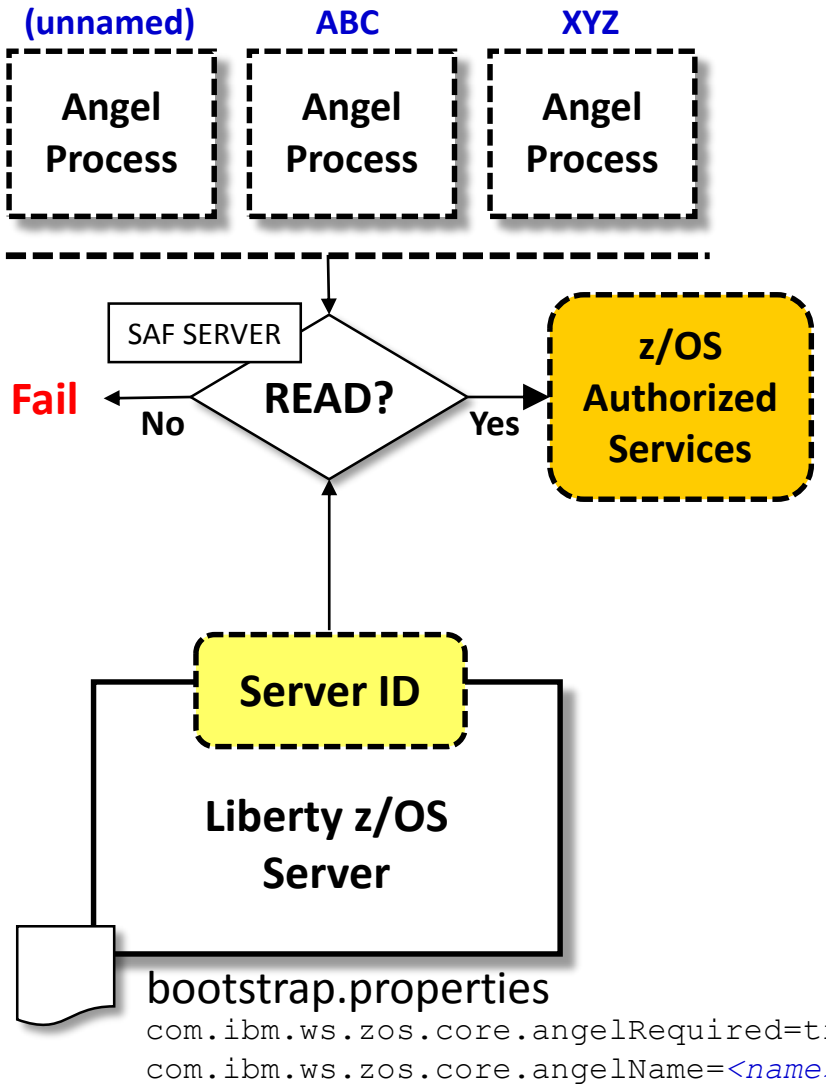
When first starting out with collectives, start small and get a feel for how they work

- Start with a single controller and single member design
- On initial test collective, start with file-based key/trust stores for SSL; later move to SAF keyrings

In general ...

- Collectives serve to organize servers that are associated with one another in some way -- test, QA, production, etc.
- If you wish to remove a server from a collective, use the "remove" function so collective understands the change

The Angel Process ... And "Named Angels" in 16.0.0.4



The Angel Process is a started task that is used to protect access to z/OS authorized services. This is done with SAF SERVER profiles.

The authorized services include: WOLA, SAF, WLM, RRS, DUMP

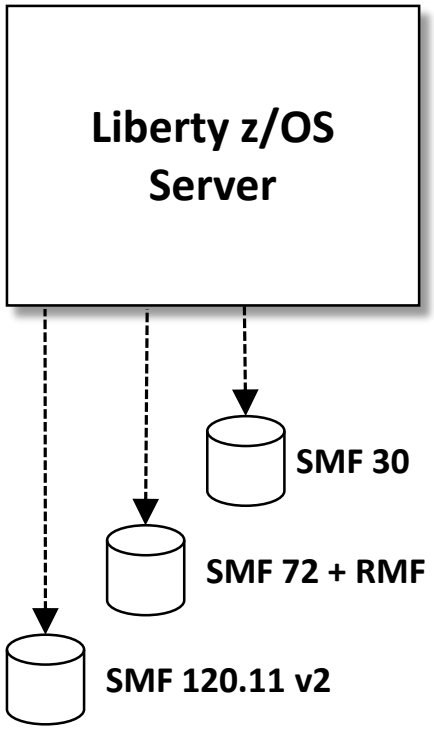
The ability to start multiple Angel processes on an LPAR was introduced in 16.0.0.4. This is called "Named Angels". It provides a way to separate Angel usage between Liberty servers:

- **Angels process can be started with a NAME='*<name>*' parameter (or it can be started as a "default" without a name). The name may be up to 54 characters.**
- **Liberty servers can be pointed at a specific Angel with bootstrap property**
- **The same SAF SERVER profile mechanism is used to protect access to authorized services (one additional SERVER profile is introduced that includes the Angel process name)**

Good practices:

- **When an "embedder" user of Liberty calls for its own named Angel, follow those instructions and set up an Angel for that product.**
- **You may create separate named Angels for isolation of Test and Production, but do not take this practice too far. A few Angels, yes; dozens, no.**
- **Establish automation routines to start the Angels at IPL**
- **Grant SAF GROUP access to the SERVER profiles, then connect server IDs as needed**

z/OS Monitoring of Liberty



Use SMF Type 30 at the STC Level

- Not very granular, but it is relatively simple

Turn on Liberty z/OS WLM Support and Use RMF Reporting

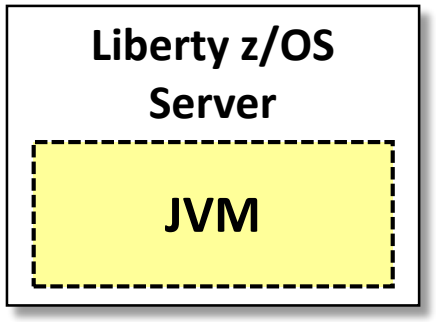
- Enable `zosWLM-1.0` feature
- Least granular = wildcard the request filter to capture all requests in one TC
- More granular = identify specific application URIs and assign to different TCs

Analyze Liberty z/OS SMF 120.11 v2 HTTP Request Records

- SMF 120.11 v2 became available with the 16.0.0.2 level of Liberty z/OS
- Details: `ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102655`
- Enable with `zosRequestLogging-1.0` feature and grant READ to BPX.SMF
- Provides detailed information about each HTTP request received in the server

Key point: you can take advantage of existing tools on z/OS to monitor Liberty z/OS

Liberty z/OS JVM Monitoring and Tuning



AdminCenter Monitoring

- If server is not part of a collective, then AdminCenter can monitor itself
- If server is part of a collective, then AdminCenter in controller can monitor other servers in the collective

Verbose GC Monitoring

- Well known practice for analyzing the activity inside a Java Virtual Machine (JVM)
- Can route to a UNIX file or to STDERR DD, which can go to JES spool
- Enable verboseGC in `jvm.options` file (`-verbose:gc`)

Other Monitoring Tools

- For example, IBM Health Center -- a client / agent model
- IBM OMEGAMON for JVMs on z/OS -- a licensed client / agent model

The good practice here is to use available tools to monitor and tune the JVM to operate optimally for the application set that runs in the JVM

Summary

Think about how you would organize Liberty z/OS servers around functional purpose -- development, test, QA, production

Think about your strategy for application hosting in Liberty compared to WAS Traditional. Many approaches; key is consistency.

Work out on paper a naming convention that ties the artifacts together

Understand the file system permission security issues and map out your strategy for file ownership. This would also imply considering "include" configuration processing.

If servers are reliant on z/OS authorized services, establish the Angel and SERVER profiles. Create automation routine to have Angel started at IPL.

Use available tools to monitor the z/OS processing (RMF, SMF) and the JVM

WebSphere Liberty 16.0.0.x Knowledge Center

http://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/as_ditamaps/was900_welcome_liberty_zos.html