WebSphere Liberty z/OS

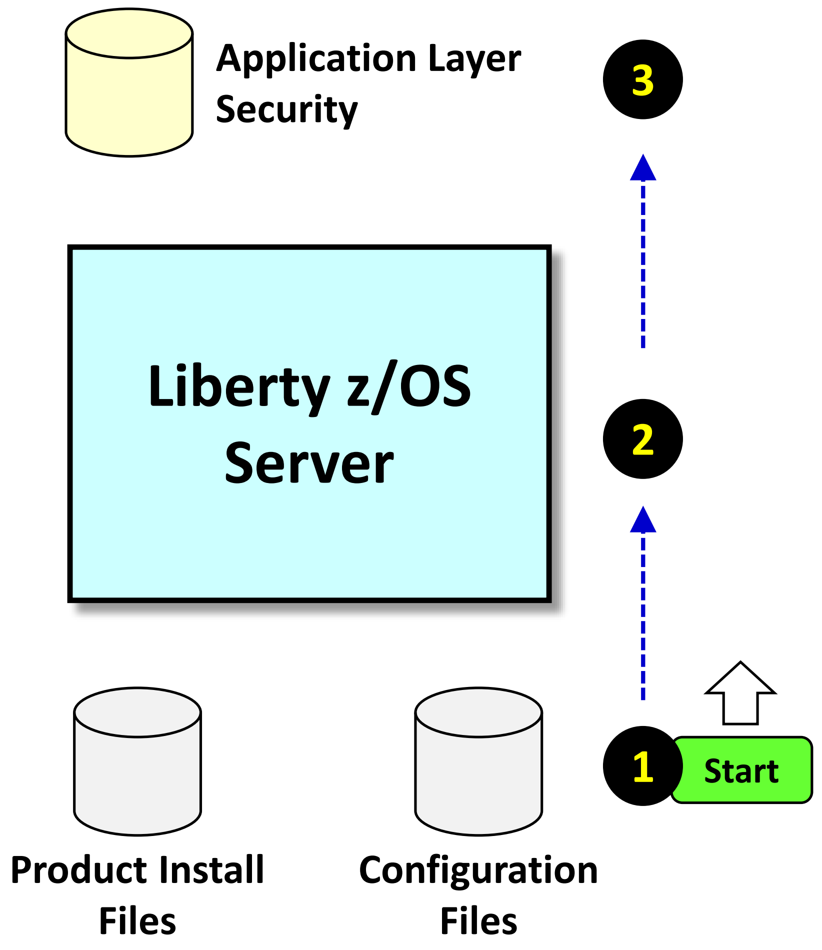
An Overview of Security

Objective of this Presentation

Provide a framework of understanding around the much larger topic of "security"

Provide a set of essential "good practices" for security

A High-Level Framework for the Security Discussion



3. **Application-related security**

   When you have an application* in a Liberty server, then quite a few more security topics surface:
   - **Encryption, and with that a discussion of digital certificates**
   - **Authentication, and with that User Registries**
   - **Role authorization enforcement**

2. **Server-related security**

   Assigning the server an identity; and allowing or restricting what that identity is capable of doing within a z/OS context

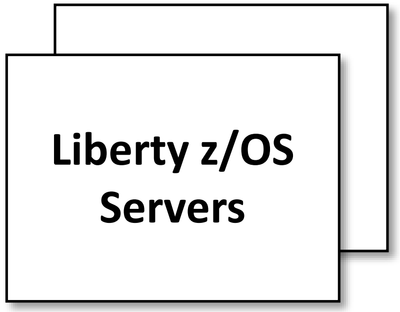
1. **File-related security**

   Involves protecting the files from unauthorized *modification*, *viewing*, and *program invocation*. This gets into file ownership and permission bits.

   Two focus areas: the install file system, and the server configuration files

* Either a user-written application, or a vendor application, or an IBM function such as the Admin Center. A server with no such application requires none of these things; but once an "application" is made available, many or all these things bubble to the surface.

General Principle: Alignment of Names of Server and Security Artifacts



Name: **T4*xxxxxx***

JCL: **T4*xxxxxx***

STARTED: **T4*xxxxxx***

IDs: **T4*xxxxxx***

Group: **T4*xxxxxx***

APPL: **T4*xxxxxx***

EJBROLE: **T4*xxxxxx.<role>***

Use a consistent prefix value for all the security artifacts

With an organized naming convention and careful deployment of security definitions, it's possible to organize Liberty servers into domains.

KnowledgeCenter `rwlp_WZSSAD_zos`

This allows for the possibility of security delegation

At a minimum this would relate to servers under a given WLP_USER_DIR; it may extend past the WLP_USER_DIR depending on your topology design

This serves two key functions:

1. **Reduces confusion (names align, easy to see relationships)**
2. **Facilitates security separation between groups of servers**

File-Related Security

Securing the install file structure; securing the configuration file structure

Review: UNIX File Permissions

	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
Bit	1	1	1	1	0	1	0	0	0
Base-2 Value	[4]	[2]	[1]	[4]	[2]	[1]	[4]	[2]	[1]
	↓	↓	↓	↓	↓	↓	↓	↓	↓

4 + 2 + 1 = **7**

4 + 0 + 1 = **5**

0 + 0 + 0 = **0**

The owner has READ, WRITE and EXECUTE

The **owner** of the file or directory

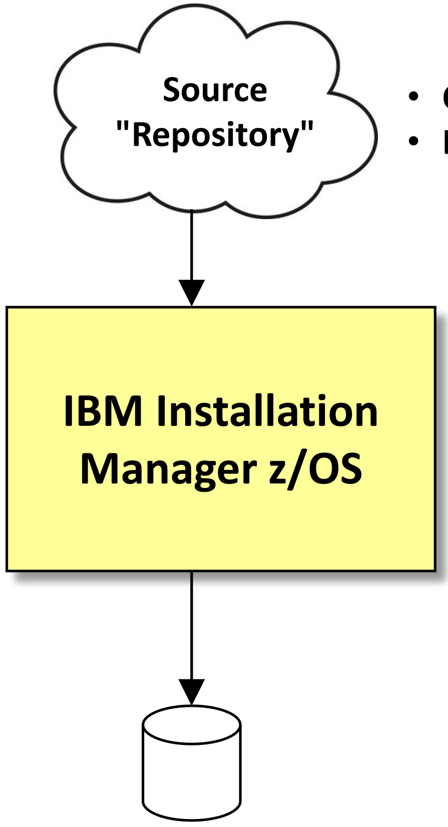
The group has READ and EXECUTE, but not WRITE

IDs that are part of the **group** for the file or directory

Others have nothing

IDs that are not the owner and not part of the group; that is, **other**

Installation Manager (IM)



File System
Mounted at location you specify

Use "Group Mode" on z/OS

- "Admin Mode" requires ID that runs IM be superuser (uid=0) ✗
- "User Mode" implies only that ID can run IM ✓
- "Group" mode allows any ID connected to the IM group to run IM ✓✓
- Use something *other than* default IMGROUP and IMADMIN

Use a "Service Zone" Concept

- It's a general good practice (provides greater flexibility)
- It allows IM install target to be R/W while users access R/O copy

"Copy out" and mount Read-Only

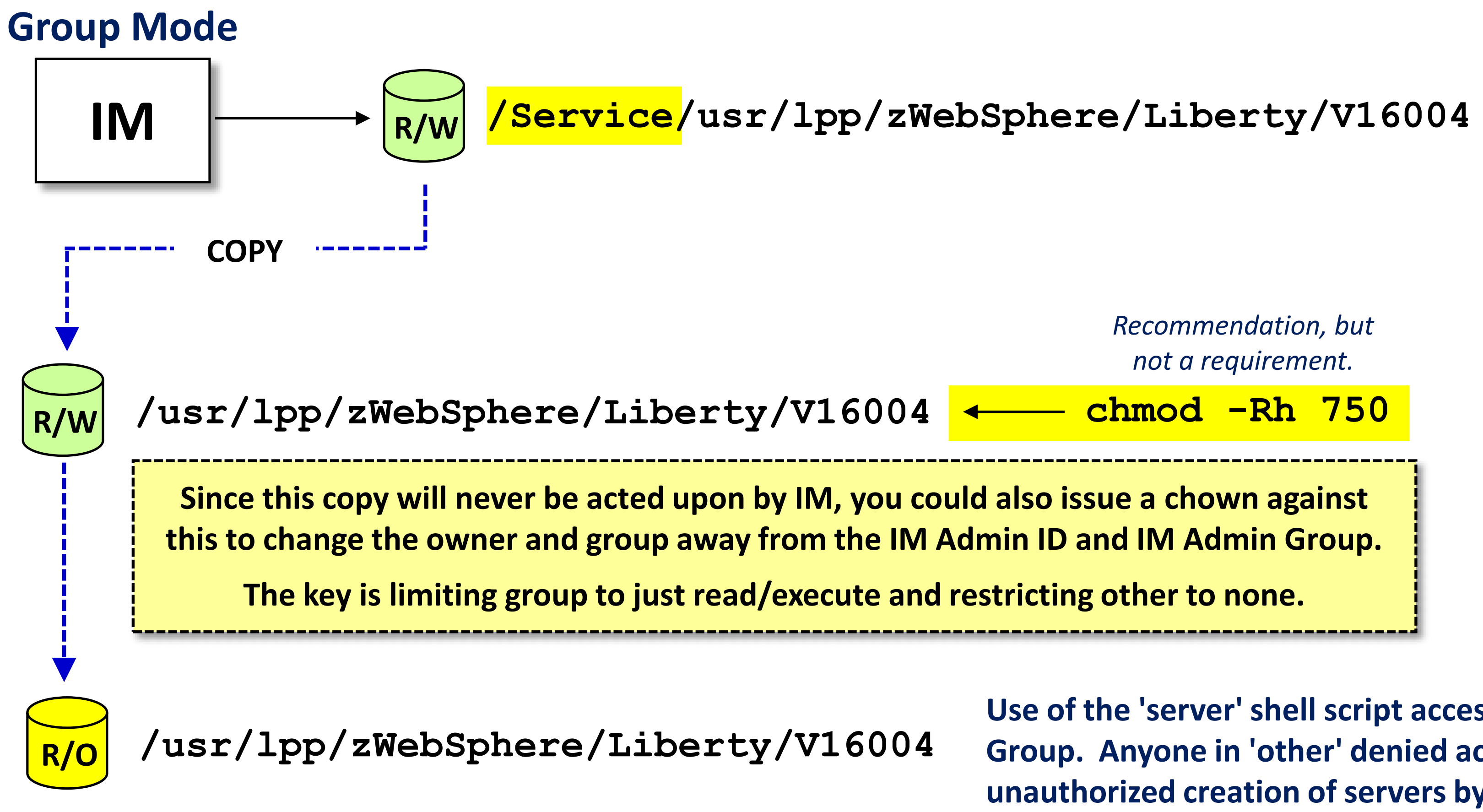
- Use standard copy tools (DFDSS COPY with RENAME)
- Consider `chmod –Rh 750` on copy to set "other" to "none"
- Mount copy as R/O

See next chart

Techdocs

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102554 **IM Guide**
http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD106391 **Sample Installation Jobs**
http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD106392 **WAS V9.0 Sample Install Jobs**

Illustration of IM Process Described on Previous Chart



`/Service/usr/lpp/zWebSphere/Liberty/V16004`

Recommendation, but not a requirement.

`/usr/lpp/zWebSphere/Liberty/V16004` ← `chmod -Rh 750`

Since this copy will never be acted upon by IM, you could also issue a chown against this to change the owner and group away from the IM Admin ID and IM Admin Group.

The key is limiting group to just read/execute and restricting other to none.

`/usr/lpp/zWebSphere/Liberty/V16004`

Many of the files are 775: write to the group and read/execute to other.

The group needs write so IM in group mode can perform updates if needed.

Other has read/execute in case you wish to make IM copy generally available to users

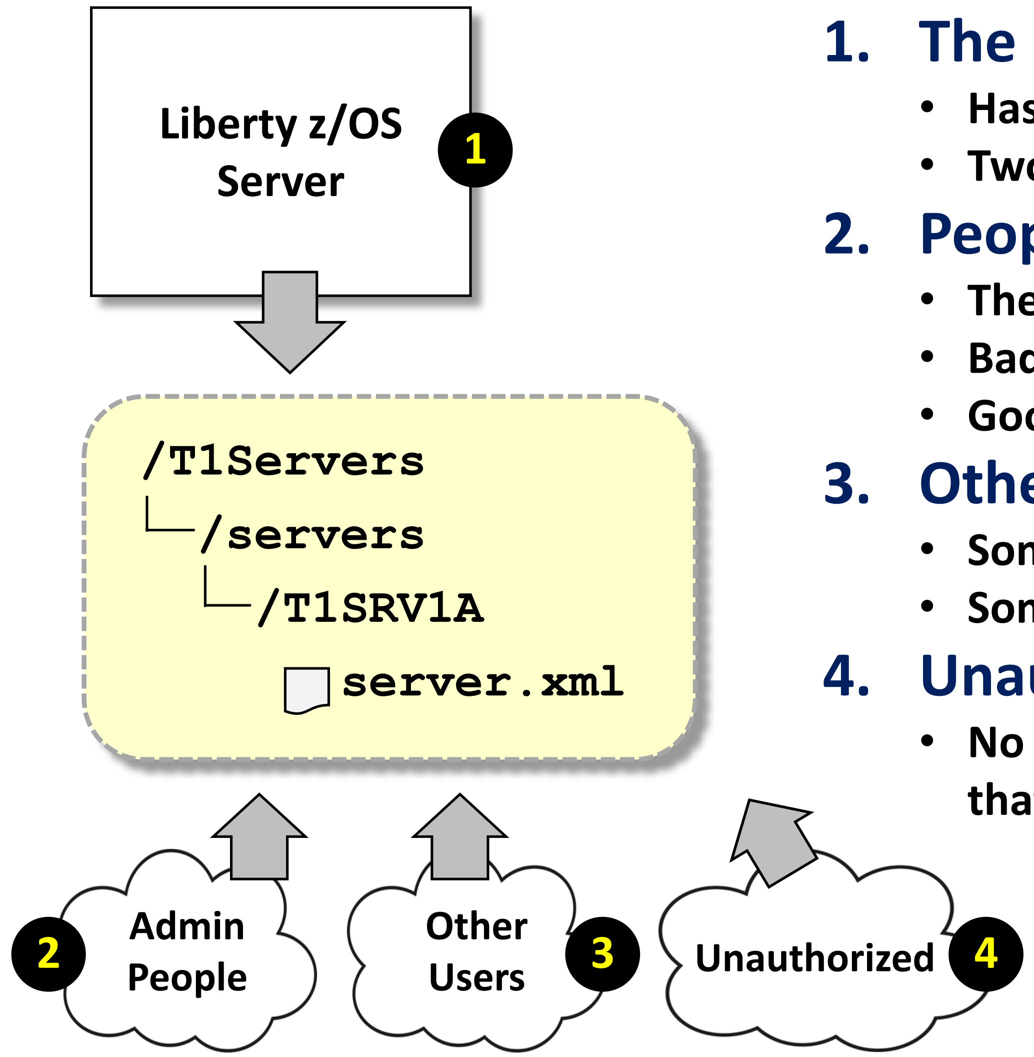
Owner (IM Admin) still has write.

Group (IM Group) reduced to read/execute

Other has no access at all

Use of the 'server' shell script accessible only to users connected to the Group. Anyone in 'other' denied access to read or execute. This prevents unauthorized creation of servers by people not connected to the group.

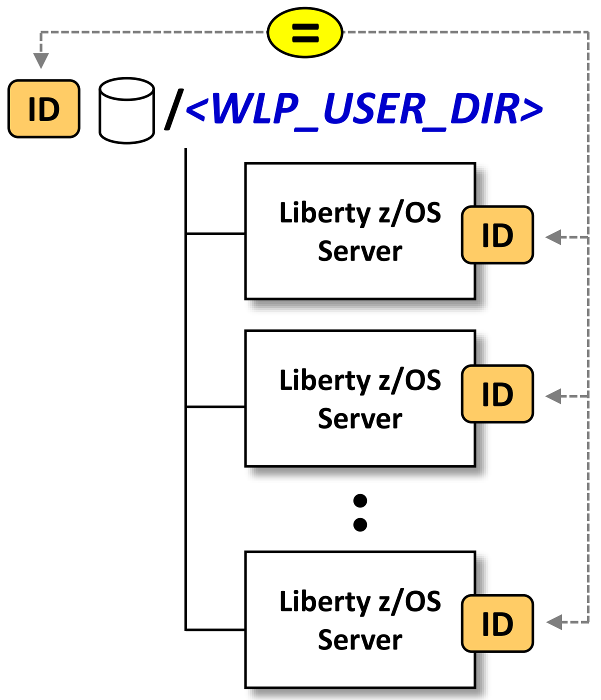
Setting the Stage: Those Seeking Access to the Configuration File Structure

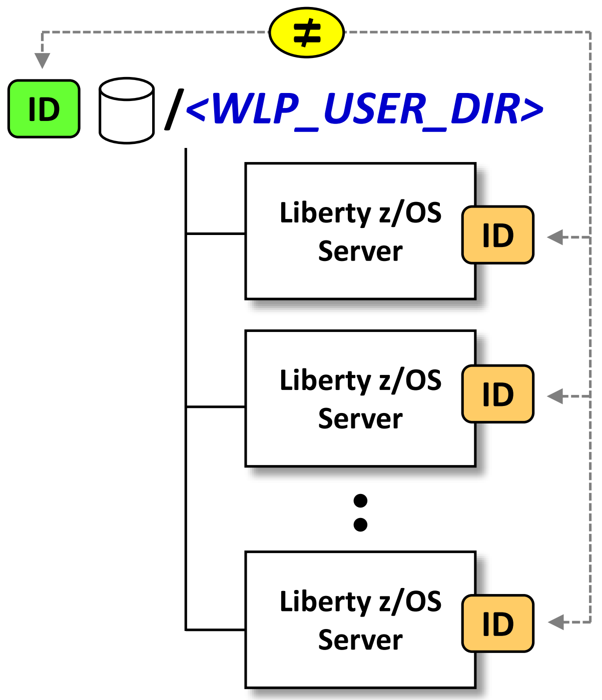


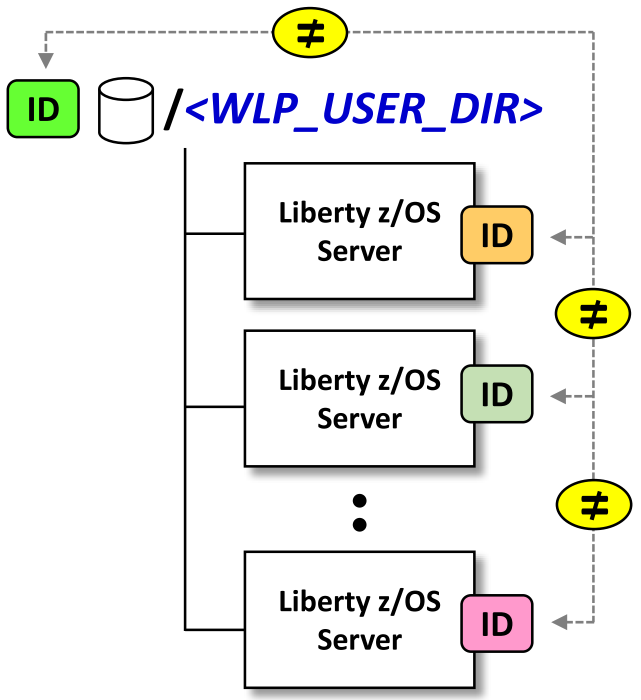
1. **The Liberty z/OS Server**
   - Has a need to both READ and WRITE
   - Two options: server ID owns files, or is a separate ID from file owner
2. **People Responsible for Administering the Server(s)**
   - They have a need to both READ and WRITE
   - Bad practice: sharing login ID/password
   - Good 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

Range of Options







- Multiple servers
- All have same STC ID
- STC ID = File Owner ID

- Multiple servers
- All have same STC ID
- STC ID ≠ File Owner ID

- Multiple servers
- Different STC IDs
- STC IDs ≠ File Owner ID

It is a matter of thee degree of identity isolation that is required

By Default ... Server Configuration Files

ID=**T1OWNER**
Group=**T1OWNG**

```
export JAVA_HOME=<path_to_64_bit_Java>
export WLP_USER_DIR=T1Servers
./server create T1SRV01
```

```
/T1Servers               750  T1OWNER T1OWNG
└ /servers               750  T1OWNER T1OWNG
   ├/.classcache         750  T1OWNER T1OWNG
   ├/.logs               750  T1OWNER T1OWNG
   ├/.pid                750  T1OWNER T1OWNG
   └/T1SRV01             750  T1OWNER T1OWNG
     ├/apps              750  T1OWNER T1OWNG
     ├/dropins           750  T1OWNER T1OWNG
     ├/logs              750  T1OWNER T1OWNG
     │└ messages.log     640  T1OWNER T1OWNG
     ├ server.xml        640  T1OWNER T1OWNG
     ├ server.env        640  T1OWNER T1OWNG
     └/workarea          750  T1OWNER T1OWNG
```

It will create the directories and files under the *<WLP_USER_DIR>* and assign ownership based on the ID and Group that created the server

This will work, but there are a few potential issues with this in a production setting:

- **If you have multiple people with a need to change configuration files, do you share the password of T1OWNER? (answer: no)**
  Sharing passwords is a very bad practice. Better to take advantage of SAF SURROGAT so permitted users can switch to the owning ID so they can make changes
- **If you have multiple people with a need to read output files, do you simply connect them to T1OWNG? (answer: no)**
  The owner group may be granted access to other SAF profiles (notably: SERVER) and you do not want others inheriting that. Better to make the configuration group be something different from the owner group and grant READ through that group.

Option: Group Ownership *Different* From File Owner Group

ID=**T1OWNER**
Group=**T1OWNG**

```
chgrp –Rh T1READG /T1Servers
```

```
/T1Servers              750   T1OWNER  T1READG
└ /servers              750   T1OWNER  T1READG
  ├ /.classcache        750   T1OWNER  T1READG
  ├ /.logs              750   T1OWNER  T1READG
  ├ /.pid               750   T1OWNER  T1READG
  └ /T1SRV01            750   T1OWNER  T1READG
    ├ /apps             750   T1OWNER  T1READG
    ├ /dropins          750   T1OWNER  T1READG
    ├ /logs             750   T1OWNER  T1READG
    │ └ messages.log    640   T1OWNER  T1READG
    ├ server.xml        640   T1OWNER  T1READG
    ├ server.env        640   T1OWNER  T1READG
    └ /workarea         750   T1OWNER  T1READG
```

People with a need to READ files are connected to this new file group

This gives you the flexibility to connect people to a group for *reading* files, but <u>not</u> have those people inherit any privileges granted to the owner group

Process:

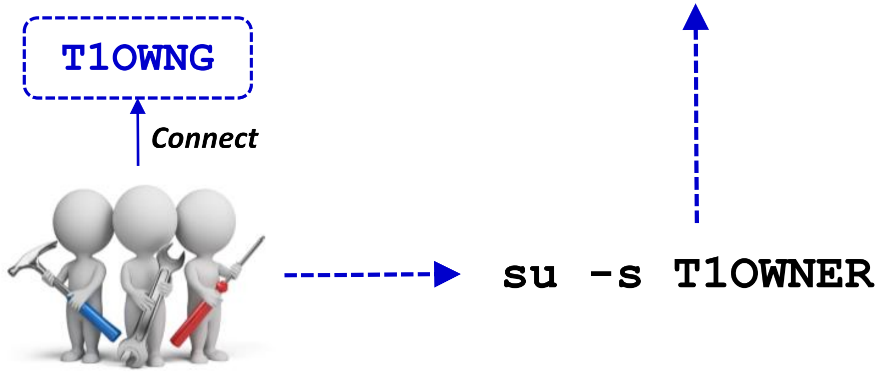
- Create the WLP_USER_DIR location
- Before creating the first server, issue (example):
  `chgrp -Rh T1READG /T1Servers`
- Create server(s). Directories and files will inherit the group from the WLP_USER_DIR information
- Connect IDs with a need to read files to the new read group; they will have ability to read but <u>not</u> to write.

Start server under T1OWNER ID*. The server will be able to write its output, and people with READ-ONLY needs can read *through* the file group.

* We'll explain how to operate the server under a separate ID in a few charts

Multiple Administrators and WRITE to Configuration Files

```
/T1Servers                750  T1OWNER T1READG
└ /servers                750  T1OWNER T1READG
   ├─/.classcache         750  T1OWNER T1READG
   ├─/.logs               750  T1OWNER T1READG
   ├─/.pid                750  T1OWNER T1READG
   └─/T1SRV01             750  T1OWNER T1READG
      ├─/apps             750  T1OWNER T1READG
      ├─/dropins          750  T1OWNER T1READG
      ├─/logs             750  T1OWNER T1READG
      │ └ messages.log    640  T1OWNER T1READG
      ├ server.xml        640  T1OWNER T1READG
      ├ server.env        640  T1OWNER T1READG
      └─/workarea         750  T1OWNER T1READG
```



Objectives:

- Avoid sharing of owning ID password between administrators
- Make owning ID have no password so it can't be used to log onto the system

Approach:

- Create owner ID with NOPASSWORD, or modify after servers are created:

```
ADDUSER T1OWNER ... NOPASSWORD     (create)
ALTUSER T1OWNER NOPASSWORD         (modify)
```

- Use SAF SURROGAT to allow users connected to the owner ID group ability to 'su' to the ID:

```
RDEFINE SURROGAT T1OWNER.SUBMIT UACC(NONE) OWNER(T1OWNG)
RDEFINE SURROGAT BPX.SRV.T1OWNER UACC(NONE) OWNER(T1OWNG)
PERMIT  T1OWNER.SUBMIT CLASS(SURROGAT) ID(T1OWNG) ACCESS(READ)
PERMIT  BPX.SRV.T1OWNER CLASS(SURROGAT) ID(T1OWNG) ACCESS(READ)
SETR RACLIST(SURROGAT) REFRESH
```

Separating the Task ID from the Configuration File Owning ID

You may have a policy that requires a started task ID not have ability to update its own configuration files. This prevents applications operating under STC ID from making malicious changes.

Liberty Started Task

Writes output under its STC ID

```
/T1Servers                 750  T1OWNER  T1READG
├─/output                  750  T1STCU   T1READG
│  ├─/.classcache          750  T1STCU   T1READG
│  ├─/.pid                 750  T1STCU   T1READG
│  └─/T1SRV01              750  T1STCU   T1READG
│     ├─/logs              750  T1STCU   T1READG
│     ├─messages.log       640  T1STCU   T1READG
│     ├─/resources         750  T1STCU   T1READG
│     └─/workarea          750  T1STCU   T1READG
└─/servers                 750  T1OWNER  T1READG
   ├─/.logs                750  T1OWNER  T1READG
   └─/T1SRV01              750  T1OWNER  T1READG
      ├─/apps              750  T1OWNER  T1READG
      ├─/dropins           750  T1OWNER  T1READG
      ├─server.xml         640  T1OWNER  T1READG
      └─server.env         640  T1OWNER  T1READG
```

Configuration File Administrators

People with need to READ files

This involves using the `WLP_OUTPUT_DIR` environment variable to direct server output to a different location

Process:

- **Add WLP_OUTPUT_DIR to `server.env` and point to a location where server output is to go**
  Example: `WLP_OUTPUT_DIR=/T1Servers/output`
- **Create that directory, and give it owner=STC ID and set the group equal to the "read group" we spoke of earlier.**
- **Connect the STC ID to the "read group" so it can *read* its configuration files. It will not have write authority.**
- **Start the server. It will create sub-directories and files under the STC ID, and the group will be inherited from the higher directory.**
- **Users connected to the "read group" will be able to read the output files.**

If STC ID is Separate, How Do We Manage Output Files under STC ID?

✗ Do not give STC ID a password

You do not want anyone to be able to log into system using the STC ID. Plus, you do not want to share ID passwords, so that ID having a password would not help when several administrators involved.

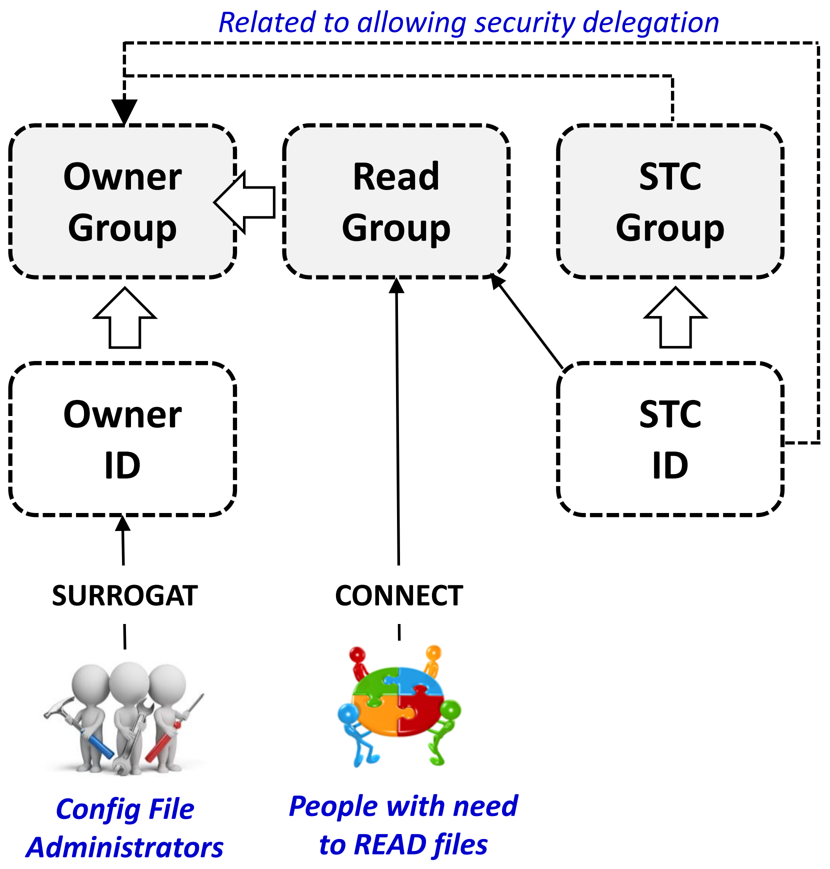
✗ Do not define SURROGAT for the Started Task ID

That would allow anyone with access to the SURROGAT to run a JVM with same authority as the server and possibly run valid business transactions.

✓ Consider granting file owning ID UNIXPRIV authority

The file owning ID will have SURROGAT defined if you followed guidance offered on previous charts. If you extend authority of the owning ID to include UNIXPRIV SUPERUSER.FILESYS.CHOWN and SUPERUSER.FILESYS.CHANGEPERMS, that ID would be able to manage files to delete, etc.

Summary: ID and Group Model for Servers

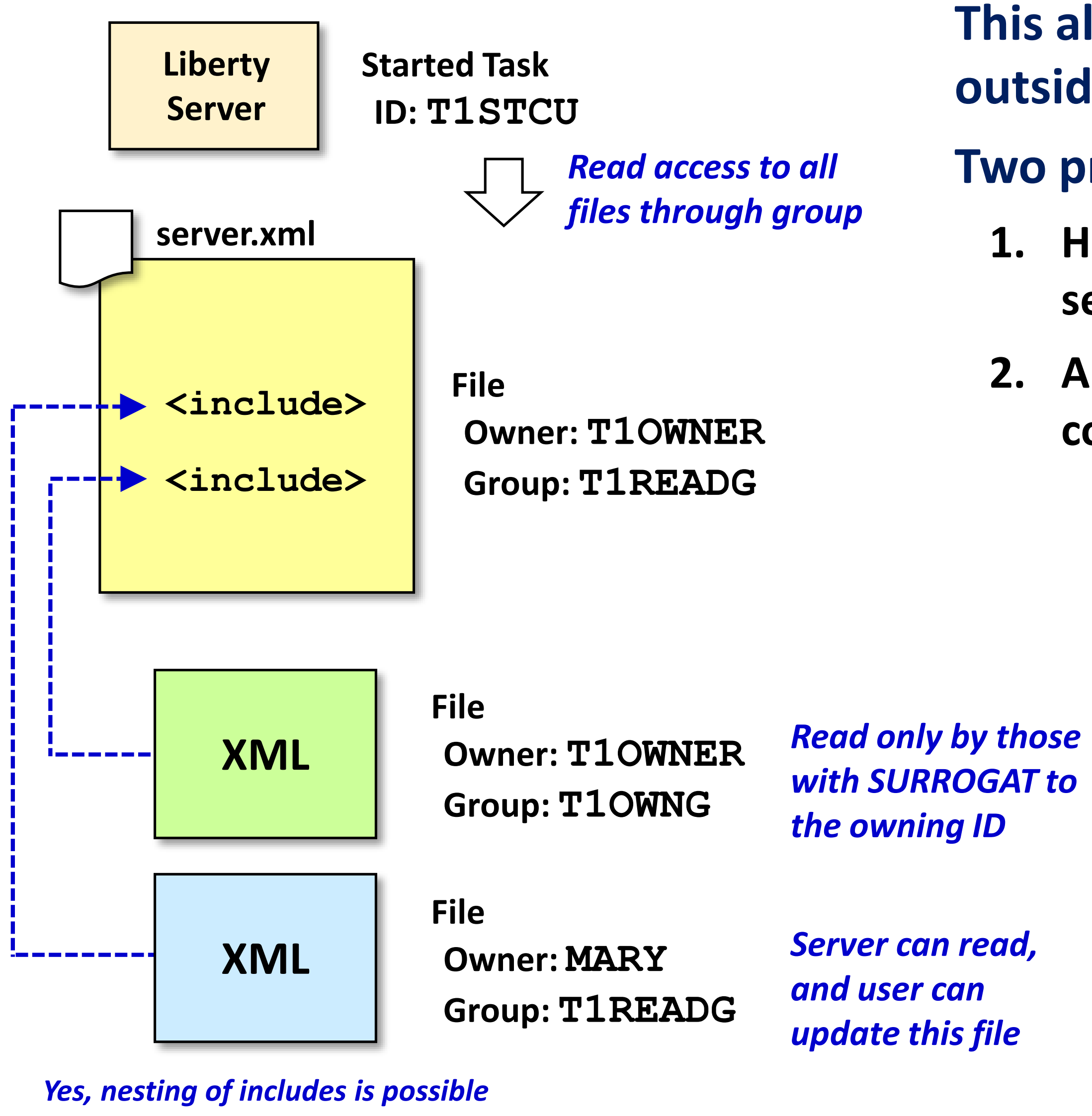

Related to allowing security delegation
Owner Group
Read Group
STC Group
Owner ID
STC ID
SURROGAT
CONNECT
Config File Administrators
People with need to READ files

Example commands:

```
ADDGROUP T1OWNG  SUPGROUP(GROUPS) OWNER(RACFADM) OMVS(AUTOGID)
ADDGROUP T1READG SUPGROUP(T1OWNG) OWNER(T1OWNG) OMVS(AUTOGID)
ADDUSER T1OWNER DFLTGRP(T1OWNG) OWNER(T1OWNG) OMVS(AUTOUID)
  HOME('/home/T1OWNER') PROGRAM('/bin/sh')) NOPASSWORD

ADDGROUP T1STCG  SUPGROUP(GROUPS)  OWNER(T1OWNG) OMVS(AUTOGID)
ADDUSER T1STCU  DFLTGRP(T1STCG)  OWNER(T1OWNG) OMVS(AUTOUID)
  HOME('/home/T1STCU') PROGRAM('/bin/sh')) NOPASSWORD

RDEFINE SURROGAT T1OWNER.SUBMIT UACC(NONE) OWNER(T1OWNG)
RDEFINE SURROGAT BPX.SRV.T1OWNER UACC(NONE) OWNER(T01OWNG)
PERMIT T11OWNER.SUBMIT CLASS(SURROGAT) ID(T1OWNG) ACCESS(READ)
PERMIT BPX.SRV.T01OWNER CLASS(SURROGAT) ID(T1OWNG) ACCESS(READ)
SETR RACLIST(SURROGAT) REFRESH

CONNECT T1STCU GROUP(T1OWNG)
CONNECT T1STCU GROUP(T1READG)

PERMIT SUPERUSER.FILESYS.CHOWN CLASS(UNIXPRIV) ID(T1OWNER) ACCESS(READ)
PERMIT SUPERUSER.FILESYS.CHANGEPERMS CLASS(UNIXPRIV) ID(T1OWNER) ACCESS(READ)
SETR RACLIST(UNIXPRIV) REFRESH
```

Review all security command examples with your security administrator prior to implementation.

"Include File" Processing



Yes, nesting of includes is possible

This allows portions of the configuration to be held in files outside the main server.xml file

Two primary uses:

1. **Hold sensitive configuration information in file that is READ to select people, but not the read group**
2. **Allow a user to update their portion of the server configuration, but not other parts of it**

For the second use-case it is important to insure the user can not override configuration in the main XML. Use the "onConflict" tag in the <include> element:

```
<include location="myIncludeFile.xml" onConflict="IGNORE"/>
```

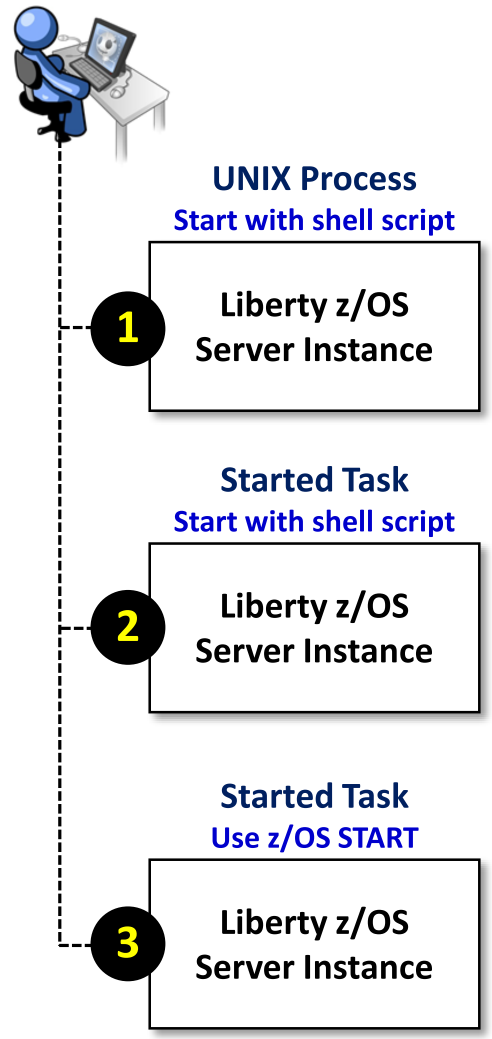
This tells Liberty to ignore XML elements in include file that are also found in the main server.xml

It does not prevent them from injecting configuration elements not found in the main server.xml. If there is a concern about that, don't use include processing.

Server-Related Security

STARTED, SERVER, and CBIND

Starting the Liberty z/OS Server



All three result in a Liberty z/OS server, and functionally there's very little difference

When started as a UNIX process, the MODIFY command interface is not present

1. **UNIX Process**
   - Use the 'server' shell script in the installation /bin directory
   - Syntax: `server start T1SRV01`
   - ID of server will be based on ID that issued the command
2. **Started Task using server shell script**
   - Set WLP_ZOS_PROCEDURE environment variable in server.env file
   - Example: `WLP_ZOS_PROCEDURE=T1PROC,JOBNAME=T1SRV01,PARMS='T1SRV01'`
   - This is how z/OS servers are started by Collective Controller
   - ID of the server will be based on the SAF STARTED profile that takes effect
3. **Started Task using START command**
   - Common proc: `START T1PROC,JOBNAME=T1SRV01,PARMS='T1SRV01'`
   - Dedicated proc: `START T1SRV01`
   - ID of the server will be based on the SAF STARTED profile that takes effect

Expectation is for production servers either #2 (via Collective Controller) or #3 will be used

Assigning ID to z/OS Started Task: SAF STARTED

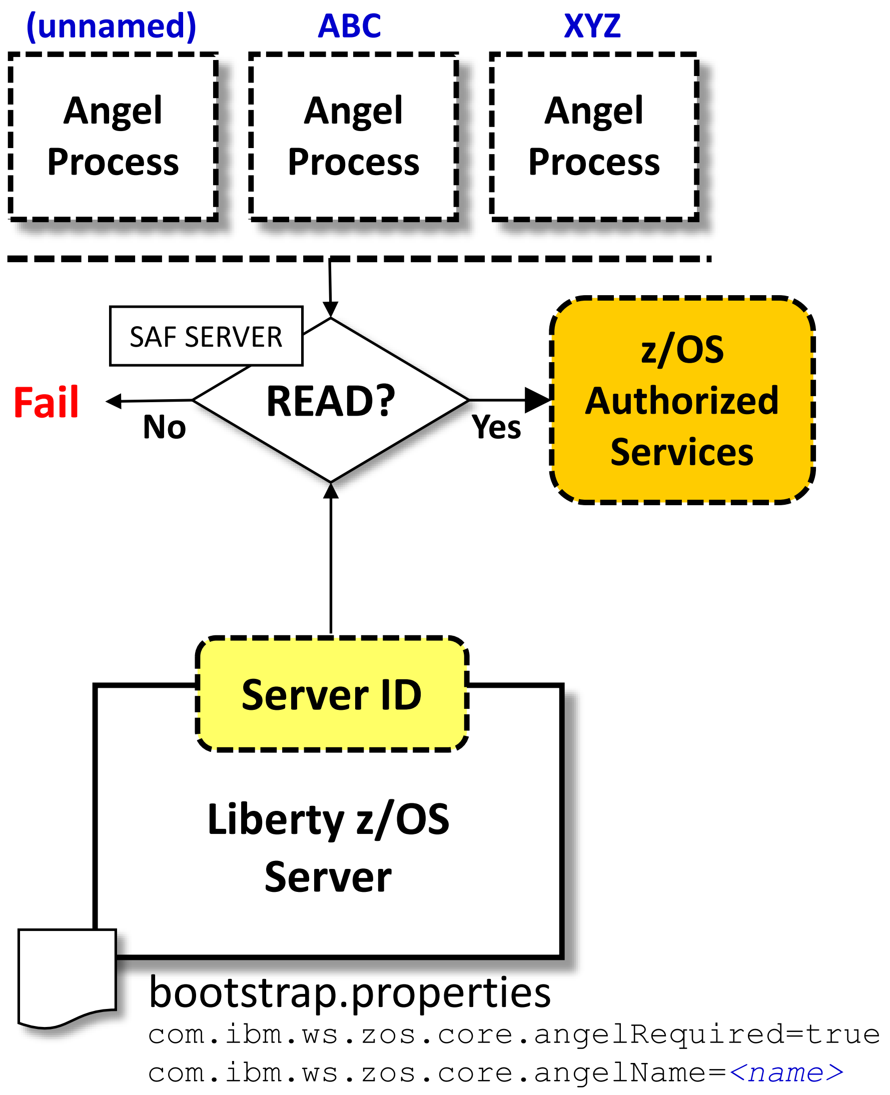
The first question here is whether you wish to have a common started task ID that is shared among servers, or if you wish each server to have a unique ID

Then the second question is whether servers under a WLP_USER_DIR will share a common JCL start proc, or use unique start procs for each server

	Common ID	*Unique IDs*
Common Proc	`START T1PROC,JOBNAME=<server>,PARMS='<server>'` `STARTED T1PROC.*`	`START T1PROC,JOBNAME=<server>,PARMS='<server>'` `STARTED T1PROC.<jobname>`
Unique Procs	`START T1SRV01` `STARTED T1*.*`	`START T1SRV01` `STARTED T1SRV01.*`

It's possible to use a combination of the above, even under the same WLP_USER_DIR. So there's no "one best answer" here. What's best is what's best for you.

Review: The Angel Process and its Role Protecting z/OS Authorized Services



```
com.ibm.ws.zos.core.angelRequired=true
com.ibm.ws.zos.core.angelName=<name>
```

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

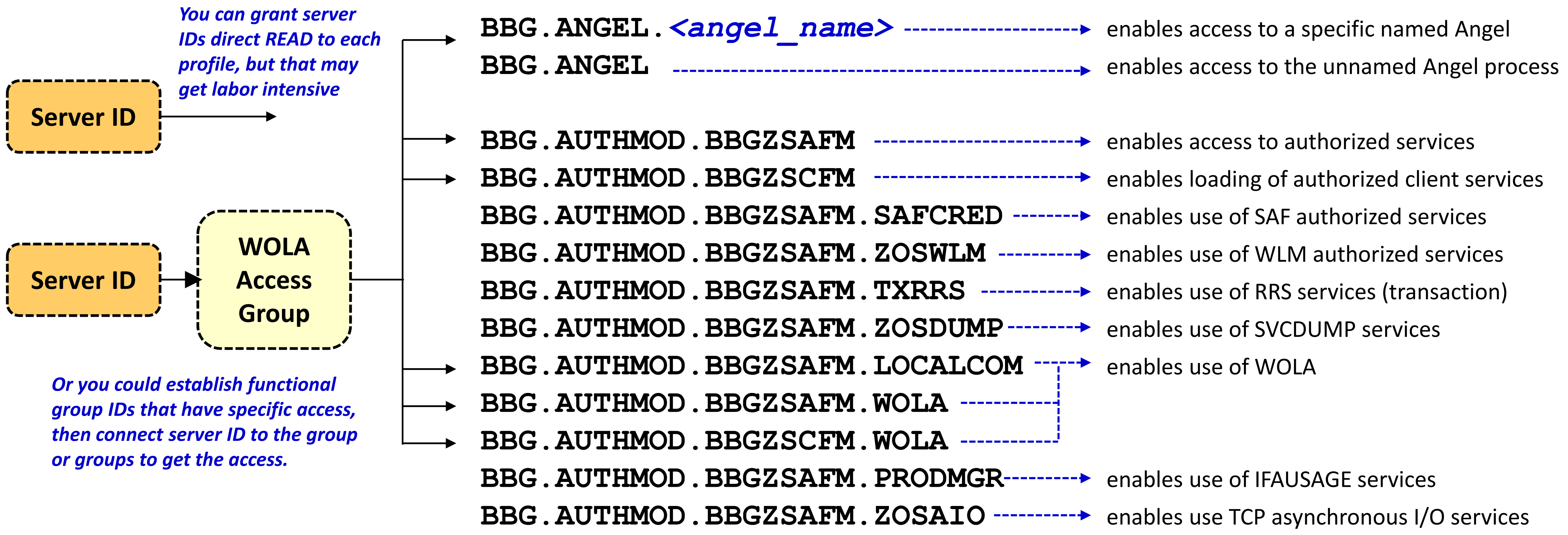
The ability to start multiple Angel processes on an LPAR was introduced in 16.0.0.4. This is called "Named Angels". It provides a way to separate Angel usage between Liberty servers:

- 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

SAF SERVER Profiles Related to the Angel Process



You can grant server IDs direct READ to each profile, but that may get labor intensive

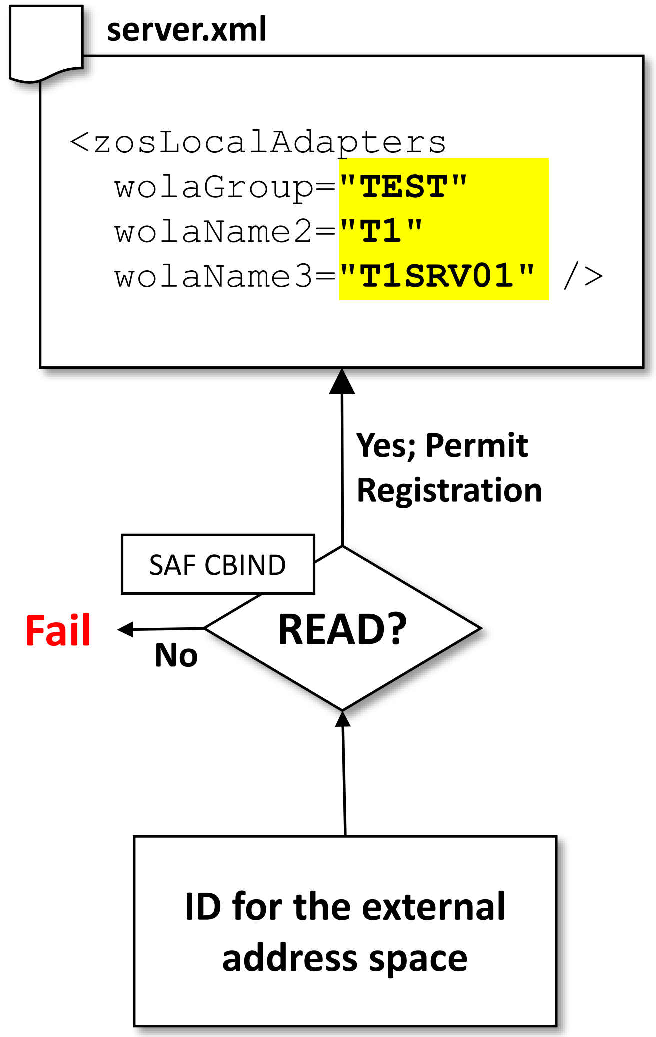
Or you could establish functional group IDs that have specific access, then connect server ID to the group or groups to get the access.

BBG.ANGEL.*<angel_name>*	enables access to a specific named Angel
BBG.ANGEL	enables access to the unnamed Angel process
BBG.AUTHMOD.BBGZSAFM	enables access to authorized services
BBG.AUTHMOD.BBGZSCFM	enables loading of authorized client services
BBG.AUTHMOD.BBGZSAFM.SAFCRED	enables use of SAF authorized services
BBG.AUTHMOD.BBGZSAFM.ZOSWLM	enables use of WLM authorized services
BBG.AUTHMOD.BBGZSAFM.TXRRS	enables use of RRS services (transaction)
BBG.AUTHMOD.BBGZSAFM.ZOSDUMP	enables use of SVCDUMP services
BBG.AUTHMOD.BBGZSAFM.LOCALCOM	enables use of WOLA
BBG.AUTHMOD.BBGZSAFM.WOLA	
BBG.AUTHMOD.BBGZSCFM.WOLA	
BBG.AUTHMOD.BBGZSAFM.PRODMGR	enables use of IFAUSAGE services
BBG.AUTHMOD.BBGZSAFM.ZOSAIO	enables use TCP asynchronous I/O services

Good practices:

- **Establish all the SERVER profiles ahead of time. Existence of profile does not grant access; READ to it does.**
- **Determine what access a server needs and grant only that; check "is available" messages in messages.log to verify**

WOLA Registration, the Three-Part Name, and the SAF CBIND Profile



```
<zosLocalAdapters
   wolaGroup="TEST"
   wolaName2="T1"
   wolaName3="T1SRV01" />
```

WOLA is a highly-efficient cross-memory mechanism

It is bidirectional: outbound from Liberty; inbound to Liberty

When an outside address space (CICS, a batch program) wants to use WOLA with an instance of Liberty, it must first build a "registration" to the Liberty address space. That is protected with a SAF CBIND profile*.

The CBIND is based on the WOLA "three-part name" the Liberty server is using, for example:

```
RDEFINE CBIND BBG.WOLA.TEST.T1.T1SRV01 UACC(NONE)
```

The ID seeking to register needs READ to that SAF CBIND. You may grant the READ directly to the ID, or grant READ to a group and connect ID to the group.

The SAF CBIND value can be wild-carded.

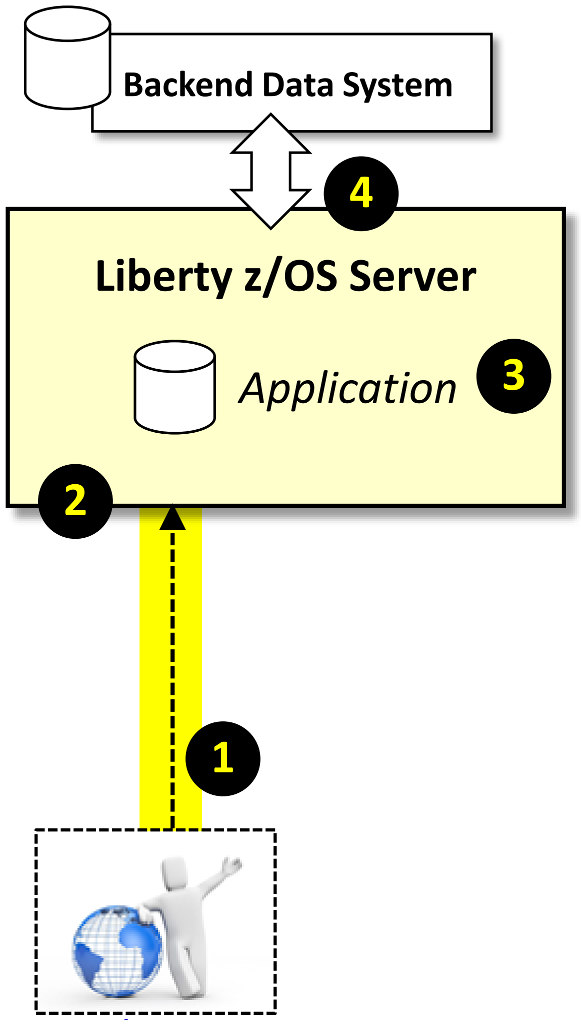
The three-part name is arbitrary, but needs to be unique on the LPAR. A good practice is to include the unique server name as the last part of the three part name, as illustrated here

* The ability of a Liberty z/OS server to use WOLA at all is controlled by that server's STC ID having READ to the appropriate SAF SERVER profiles. The SAF CBIND then controls which outside address spaces can "register into" the Liberty z/OS server.

Application-Related Security

Encryption, Authentication, Authorization

The Presence of an Application* Triggers Several More Security Topics



1. **Transport Layer Encryption**
   Depending on the client access method, encrypting the data link may be required. This is where a discussion of "Transport Layer Security" (TLS, commonly referred to as "SSL") comes up. This involves certificates and key/trust stores. On z/OS that can be managed in SAF.

2. **Authentication**
   A client presents itself to the server and claims to be "Person X." Are they who they claim to be? That is authentication. There are many forms, from basic ID/password to more sophisticated third-party authentication and the passing of identity tokens.

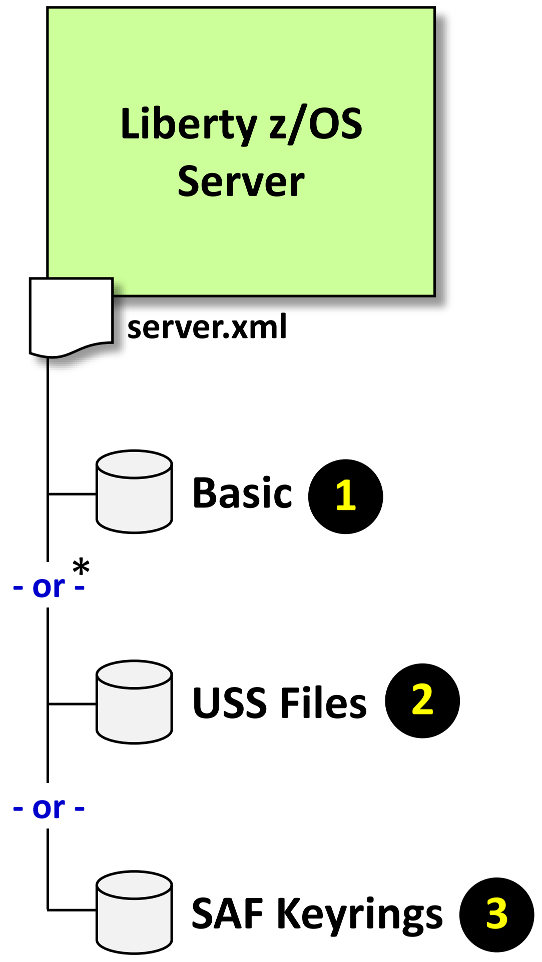
   A closely related topic is "User Registries," which is where information about users is kept. Again, on z/OS that can be SAF.

3. **Application Role Authorization**
   Once authenticated, an application may enforce different levels of authority the client is permitted to have. That is done using "roles," and role authorization involves validating which role the authenticated user belongs to. On z/OS, this can be done in SAF.

4. **Backend System Access**
   If the application reached back to a data system, what ID does it use? The backend data system may allow or disallow access based on the ID that is presented. This is a somewhat complex topic as the ID that's presented has several "it depends" qualifiers. We defer this to another presentation.

* An "application" could be one you wrote, or a vendor application, or an IBM function such as the Liberty AdminCenter. The point is, it is software function a client communicates with.

TLS (aka "SSL") and Digital Certificates



* Combinations within a given Liberty z/OS server is possible.

1. Basic Key and Trust Store

- Simple one-line addition to the server.xml

```
<keyStore id="defaultKeyStore" password="Liberty"/>
```

- Satisfies *basic* requirements for TLS, but good only for initial validation. Not good for testing (self-signed certificate), and certainly *not* for production.

2. File-based Key and Trust Store

- Same mechanism as used on distributed platforms (keytool or ikeyman)
- Can be used for testing and production
- File password in server.xml can be encoded. SAF keyrings eliminate need for passwords

3. SAF-based Keyrings

- The server.xml file points to SAF as its key and trust store
- Use SAF keyrings to hold digital certificates and signer (CA) certificates
- No passwords in server.xml
- Access to SAF keyrings protected by SAF IRR.DIGTCERT.* profiles
- General good practice to use z/OS facilities when on z/OS

SAF Keyring Support in server.xml

```
<feature>ssl-1.0</feature>  1

<sslDefault sslRef="DefaultSSLSettings" />  2

<ssl id="DefaultSSLSettings"  3
  keyStoreRef="DefaultKeyStore" trustStoreRef="DefaultTrustStore" />
<keyStore id="DefaultKeyStore" location="safkeyring://Keyring.LIBERTY"
  password="password" type="JCERACFKS" fileBased="false" readOnly="true" />
<keyStore id="DefaultTrustStore" location="safkeyring:///Keyring.LIBERTY"
  password="password" type="JCERACFKS" fileBased="false" readOnly="true" />
          4

<ssl id="T1SSLConfig"  5
  keyStoreRef="T1KeyStore" trustStoreRef="T1TrustStore" />  6
<keyStore id="T1KeyStore" location="safkeyring:///Keyring.T1"
  password="password" type="JCERACFKS" fileBased="false" readOnly="true" />
<keyStore id="T1TrustStore" location="safkeyring:///Keyring.T1"
  password="password" type="JCERACFKS" fileBased="false" readOnly="true" />

<!- HTTP using non-default SSL config -->
 <httpEndpoint id="defaultHttpEndpoint"
     host="*"
     httpPort="9080"
     httpsPort="9443" >
     <sslOptions sslRef=“T1SSLConfig" />  7
 </httpEndpoint>
```

1. **Update <featureManager> list**
   The `ssl-1.0` feature enables the support to use SAF for the key/trust stores. It may be auto-loaded by other features, but specifying it explicitly is a good practice.
2. **Specify default SSL settings**
   The <sslDefault> tag specifies the default SSL settings for the server. If you have multiple SSL settings, you definitely need this. A good practice to specify in all cases.
3. **Default SSL settings**
   You may customize and have the "default" be tailored to your server. Or you may retain a true "default" and provide a separate customized SSL settings (block #5).
4. **The "password" for SAF keyrings**
   SAF does not use a password, but the Liberty keystore code requires it. This is just a dummy placeholder.
5. **Specific SSL settings**
   If you wish, you can provide SSL settings specific to your server and reference it from the HTTP endpoint (block #7)
6. **Naming convention prefix in keyring name**
   Whether default or specific SSL settings, it's a good practice to have the keyrings used by the server reference the naming convention prefix for the server.
7. **Specifying SSL options for HTTP endpoints**
   If you wish a set of HTTP endpoints to use something other than the default SSL settings, point to the SSL options using the tag shown here.

Enabling Crypto Hardware Support for Liberty z/OS

If you have crypto hardware on the System z machine, take advantage of it:

```
/T1Servers
└─/servers
  └─/T1SRV01
     ├── java.security
     ├── server.xml
     └── jvm.options
```

Copy java.security file to Liberty z/OS server configuration directory and update

- Copy java.security file from the /lib/security directory of your 64-bit Java SDK installation
- Update as shown here:

```
security.provider.1=com.ibm.crypto.ibmjcehybrid.provider.IBMJCEHYBRID
security.provider.2=com.ibm.crypto.hdwrCCA.provider.IBMJCECCA
security.provider.3=com.ibm.jsse2.IBMJSSEProvider2
security.provider.4=com.ibm.crypto.provider.IBMJCE
security.provider.5=com.ibm.security.jgss.IBMJGSSProvider
security.provider.6=com.ibm.security.cert.IBMCertPath
security.provider.7=com.ibm.security.sasl.IBMSASL
security.provider.8=com.ibm.xml.crypto.IBMXMLCryptoProvider
security.provider.9=com.ibm.xml.enc.IBMXMLEncProvider
security.provider.10=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
security.provider.11=sun.security.provider.Sun
```

Update server.xml SSL settings and update type= value:

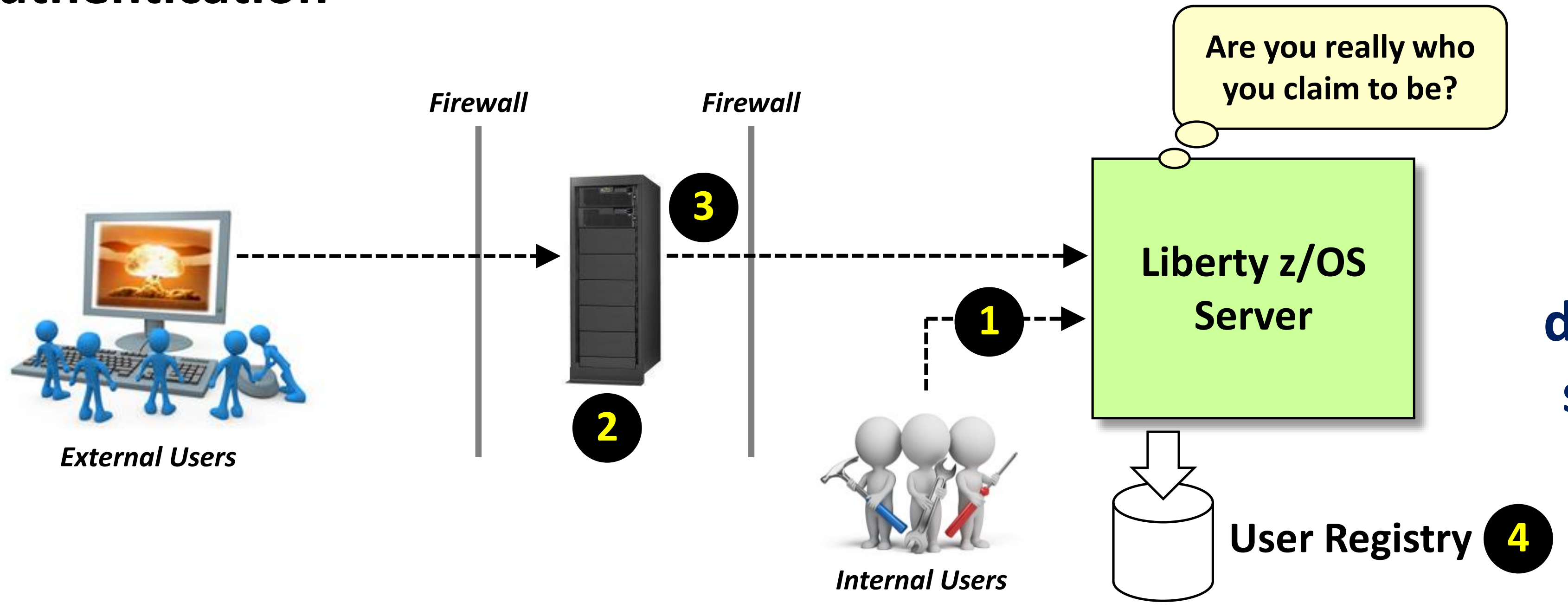
```
<keyStore id="CellDefaultKeyStore" location="safkeyring:///Keyring.T1"
   password="password" type="JCEHYBRIDRACFKS" fileBased="false" readOnly="true" />
<keyStore id="CellDefaultTrustStore" location="safkeyring:///Keyring.T1""
   password="password" type="JCEHYBRIDRACFKS" fileBased="false" readOnly="true" />
```

Create jvm.options file and point to java.security file to use

```
-Djava.security.properties==/T1Servers/servers/T1SRV01/java.security
```

Techdocs http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101213

Authentication



This is a big topic, and the implementation details go far beyond the scope of this document

1. **UserID / Password**
   What most think of when "authentication" is discussed. This is an option with Liberty, and may be suitable, depending on the application architecture requirements.
2. **Single Sign-On / Third-Party Auth**
   This includes technologies such as LTPA tokens, SPNEGO, OpenID, SAML, JSON Web Tokens (JWT). This is very common when the user-base is very large.
3. **Client Certificate Authentication**
   This is sometimes referred to as "two-way SSL," it involves authenticating using TLS encryption certificates. This is often used to authenticate a programmatic layer, such as a proxy server.
4. **User Registry**
   Any discussion of authentication eventually leads to the topic of where user information is held, and that's a "user registry." Typical options are LDAP and SAF.

Liberty z/OS User Registries

User Registry Options with Liberty z/OS:

✗ **Basic**

- The user ID and password values are maintained in the server.xml (or an include file)
- Adequate for initial validation and *some* testing, but not for readiness testing, QA, or production use.

✓ **SAF**

- The user ID and password values are maintained in SAF
- Very secure and very well-suited for production
- Can be an issue if the user-population is very large, or may self-register

This topic brings a few more SAF-related requirements to the table, so we'll explore this a bit more ...

More

✓ **LDAP**

- Liberty z/OS access an LDAP server (on z/OS or remote)
- Commonly used when the user-population is large and dynamic
- Good practice: maintain bind password in a separate "include" file, not in server.xml

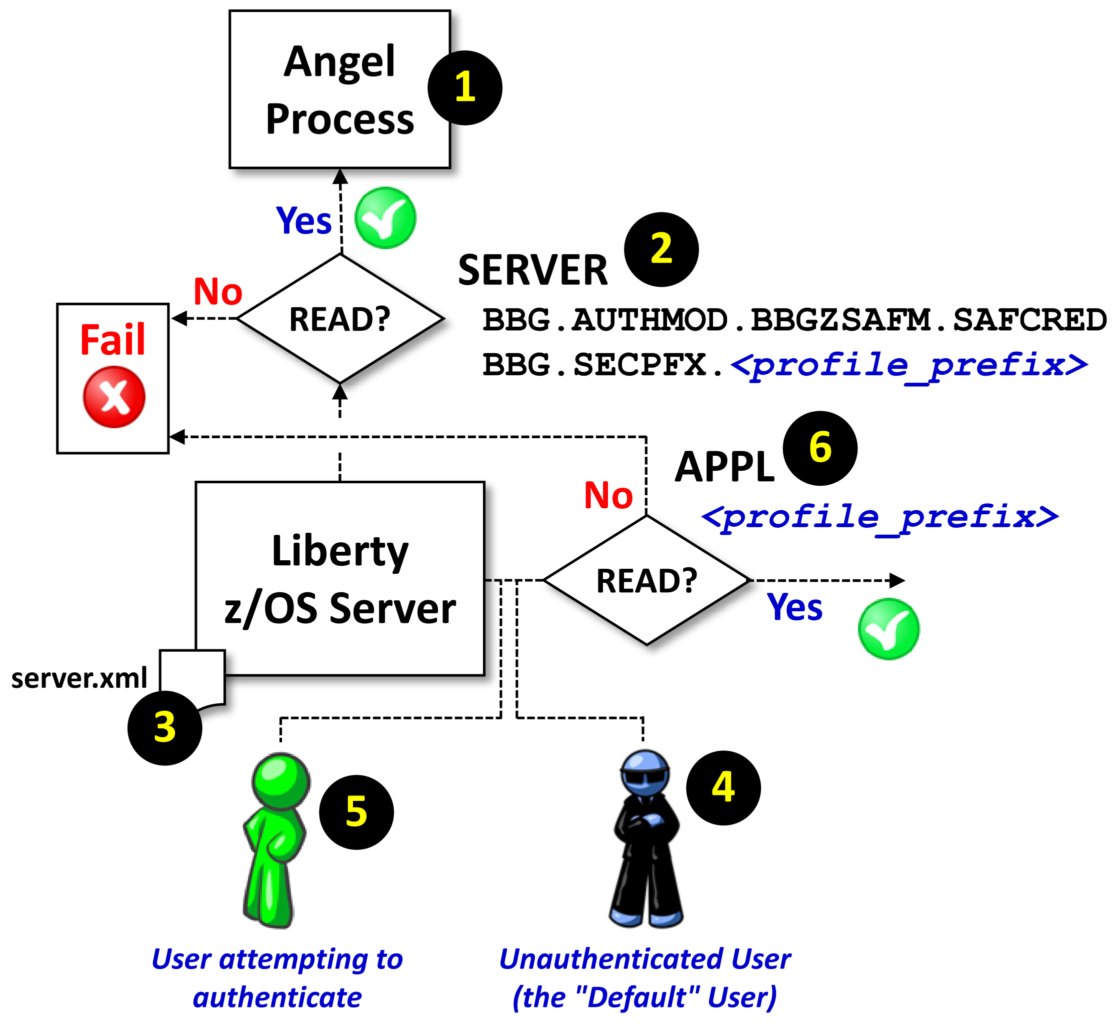
✓ **Federated Registries**

- Multiple registries are employed: LDAP and SAF typically
- Often involves "distributed identity mapping" -- mapping an LDAP user to a SAF user

⚠ **Custom**

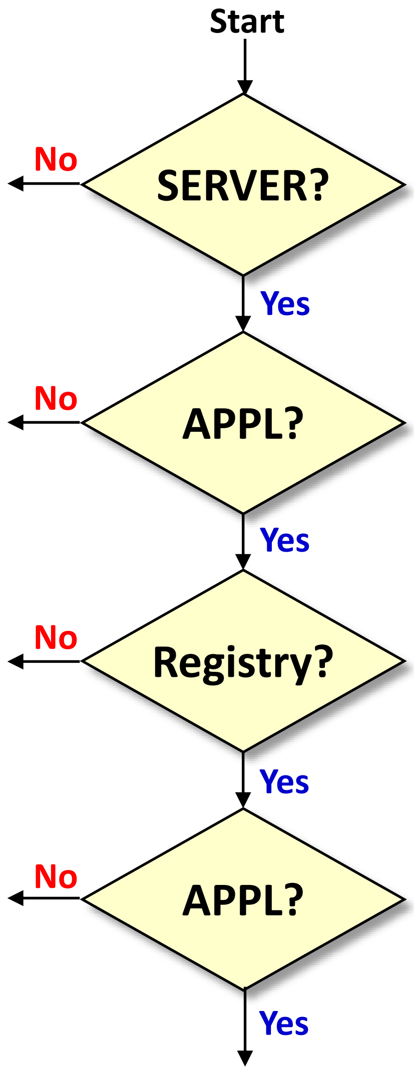
- Use the `UserRegistry` class to implement a custom registry
- Should be thoroughly reviewed before used in anything other than development and test

To Use SAF as a Registry Requires a Few Things ...


Angel Process
1
Yes
SERVER
2
No
READ?
Fail
BBG.AUTHMOD.BBGZSAFM.SAFCRED
BBG.SECPFX.<profile_prefix>
APPL
6
No
<profile_prefix>
Liberty z/OS Server
READ?
Yes
server.xml
3
5
4
User attempting to authenticate
Unauthenticated User (the "Default" User)

1. **Angel Process available to the server**
   - Either an unnamed Angel or a named Angel
2. **SERVER profiles with the server ID having READ**
   - BBG.AUTHMOD.BBGZSAFM.SAFCRED with server ID = READ
   - BBG.SECPFX.*<profile_prefix>* where the prefix value is related to your server prefix, for example T1
   - Server ID granted READ to this SECPFX profile
3. **The server.xml specifies SAF and names prefix value**
   - This involves a few lines of XML will show you in a chart or two
4. **A defined "unauthenticated" (i.e. "default") user**
   - This is the ID that is used prior to successful authentication
   - This ID should have no TSO segment, and be RESTRICTED
5. **User authenticating must have valid SAF definition**
   - The user attempting to authenticate must have a valid SAF definition (OMVS segment, valid home directory, not revoked)
6. **APPL profile with READ to required IDs**
   - The APPL profile is equal to the *<profile_prefix>* value you defined on the SERVER profile (#2)
   - The server ID has READ to this APPL
   - The unauthenticated user has READ to this APPL
   - The ID attempting to authenticate has READ to this APPL

Or, to Illustrate Another Way ...



SAF Registry requires access to z/OS authorized services, which means the server needs access to the SERVER profiles protected by an Angel

At server startup the server will check to see if (a) an unauthenticated user is defined, and (b) it has access to an APPL profile equal to the profile prefix specified

The first user asking to be authenticated shows up. The server checks with SAF to see that the ID presented is valid

The server then checks to see if the authenticated ID has access to the APPL profile, which is what gives it permission to access the applications in the server

The user is authenticated ... the next step may be to check application role authority

Updates to server.xml in Support of SAF Authentication

server.xml

```
<feature>zosSecurity-1.0</feature>
<feature>appSecurity-2.0</feature>                        1

<safRegistry id="saf" realm="MYPLEX" />                   2

<safCredentials
  unauthenticatedUser="T1DFLTU"                           3
  profilePrefix="T1" />

<safAuthorization id="saf"
  racRouteLog="ASIS"
  enableDelegation="true" />
                                                          4
<safRoleMapper
  profilePattern="%profilePrefix%.%resource%.%role%"
  toUpperCase="false" />
```

1. Liberty Features

zosSecurity-1.0 enables SAF authentication and authorization.

appSecurity-2.0 enables application security, including role checking and RunAS delegation.

2. <safRegistry>

This enables SAF-access function.

3. <safCredentials>

Here the "unauthenticated user" is named, and the "profile prefix" is specified. The unauthenticated user should be a RESTRICTED ID (meaning: it gains no access via group or via RACF global entry checking.

The profile prefix should be equal to whatever "starting prefix" you choose for your naming convention. An APPL profile is created to match this value.

4. Authorization XML

These final two sections of XML relate to application role authorization. We'll discuss this in an upcoming chart.

Example RACF Setup Definitions for Unauthenticated User and APPL

```
RDEFINE SERVER BBG.SECPFX.T1 UACC(NONE)                                  1
PERMIT BBG.SECPFX.T1 CLASS(SERVER) ACCESS(READ) ID(T1STCU)

ADDGROUP T1DEFG SUPGROUP(T1OWNG) OWNER(T1OWNG) OMVS(AUTOGID)
                DATA('T1 unauthenticated user group')
2
ADDUSER  T1DEFU DFLTGRP(T1lDEFG) OWNER(T1OWNG)
                NAME('T1 default user') OMVS(AUTOUID) HOME(/home/T1DEFU)
                PROGRAM(/bin/sh))
                DATA("RACF ADMIN: DO NOT PERMIT THIS USER TO RESOURCE PROFILES")
                NOPASSWORD NOOIDCARD RESTRICTED

ADDGROUP T1USRG SUPGROUP(T1OWNG) OWNER(T1OWNG) OMVS(AUTOGID)  3
                DATA('T1 authenticated user group')

RDEFINE APPL T1 UACC(NONE)  DATA('Controls access to T1 servers')  4

PERMIT       T1 CLASS(APPL) ID(T1STCU) ACCESS(READ)
PERMIT       T1 CLASS(APPL) ID(T1DEFU) ACCESS(READ)   5
PERMIT       T1 CLASS(APPL) ID(T1USERG) ACCESS(READ)
```

Review all security command examples with your security administrator prior to implementation.

1. **SERVER Class Prefix**
   This defines the security prefix, which is later matched with an APPL profile, and is used to pre-pend EJBROLE definitions. The server ID is granted READ.
2. **Unauthenticated Group/User**
   The "default" user and its group. Note the RESTRICTED specified on the user. This ID is deliberately very low authority.
3. **User Group for APPL Access**
   IDs attempting to authenticate will need READ to the APPL. You could UACC(READ) the APPL profile, or you can create an access group and grant IDs to this group to gain access to the APPL profile.
4. **APPL Profile**
   This matches the profile prefix SERVER profile, and aligns with your naming convention.
5. **Permit Users to APPL**
   The server ID and the default ID are granted access, as well as the user group which allows other IDs access.

Application "Roles"

Who Are You?

This is ***authentication***. We looked at that earlier. Once a user successfully authenticates, the server knows who the user is.

What Are You Allowed To Do?

This is ***authorization***. It is a function of the application. An application may or may not define different "roles," but if roles are defined, then the server gets involved to help the application determine if a user is a member of the defined role.

Application WAR File web.xml

```
<servlet>
   <servlet-name>myHello</servlet-name>
   <servlet-class>HelloServlet</servlet-class>
   <security-role>
      <role-name>MyRole</role-name>
   </security-role>
</servlet>
```

Application "roles" are used to define what a user is allowed to do (that is "authorization").

An application is not required to define roles, but if it does then a mechanism is needed to check whether the authenticated ID is permitted access to the role:

- **Basic (in server.xml)**
- **SAF (in EJBROLE)**

Role Validation … Basic vs. SAF

server.xml

```
<feature>zosSecurity-1.0</feature>
<feature>appSecurity-2.0</feature>

<safRegistry id="saf" realm="MYPLEX" />

<safCredentials
  unauthenticatedUser="T1DFLTU"
  profilePrefix="T1" />

<application
   id="myServlet" name="myServletWAR" type="war"
   location="/<path>/myServletWAR.war" >
   <application-bnd>
      <security-role name="MyRole">
        <group name="T1USRG" />
      </security-role>
   </application-bnd>
</application>
```

If the authenticated user is a member of this group, then they have access to that role.

server.xml

```
<feature>zosSecurity-1.0</feature>
<feature>appSecurity-2.0</feature>

<safRegistry id="saf" realm="MYPLEX" />

<safCredentials
  unauthenticatedUser="T1DFLTU"
  profilePrefix="T1" />

<application
   id="myServlet" name="myServletWAR" type="war"
   location="/<path>/myServletWAR.war" >
</application>

<safAuthorization id="saf"
  racRouteLog="ASIS" />

<safRoleMapper
  profilePattern="%profilePrefix%.%resource%.%role%"
  toUpperCase="false" />
```

Go to SAF for EJBROLE

ASIS = Log as specified in RACF profile

The %resource% element is often not used, leaving just prefix + role

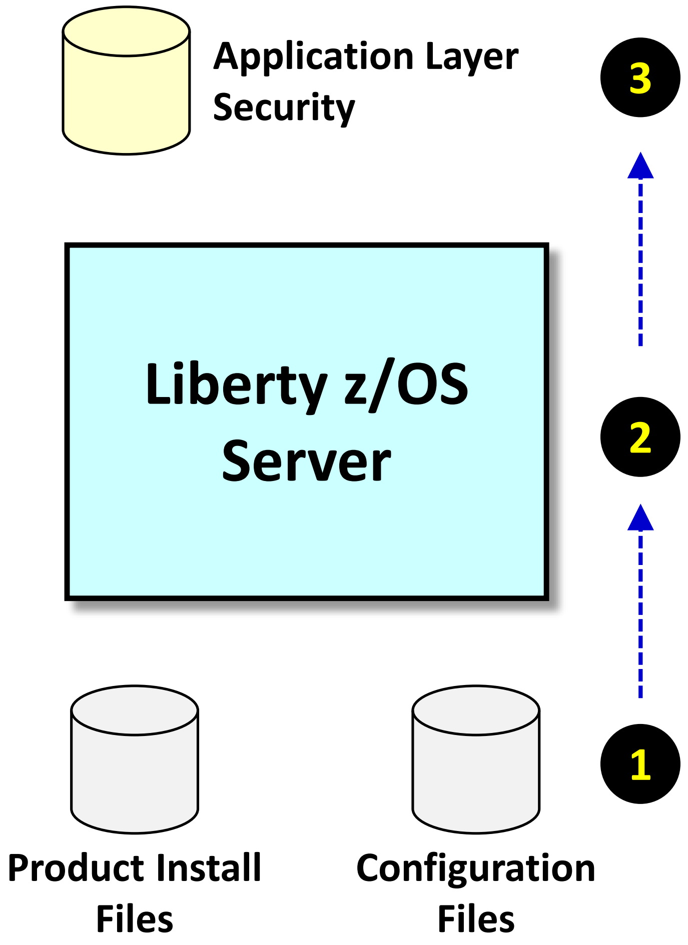
SAF

```
RDEFINE EJBROLE T1.myServletWAR.MyRole UACC(NONE)
PERMIT T1.myServletWAR.MyRole CLASS(EJBROLE)
  ID(T1USRG) ACCESS(READ)
```

Security Topics for Other Presentations ...

- **LDAP as user registry ... related: federated registries**
  - Common practice with distributed platform Liberty
  - Can be done with Liberty z/OS, with LDAP on a distributed platform or on z/OS
  - LDAP ID can be mapped to a SAF ID
- **Single sign-on / third party authentication**
  - Common practice when the application has thousands (or millions) or external users
  - Authentication is done elsewhere a token representing the user is flowed back to Liberty z/OS
  - Examples: LTPA tokens, SPNEGO, OpenID, SAML, JSON Web Tokens (JWT)
- **Backend Data access security**
  - Essential question is: what identity is presented to the backend data resource manager?
  - The answer is "it depends" ... on the resource adapter used and how the connectivity is designed.
  - Can be an "alias" (hard-coded ID/PW); it can be a designated functional ID (runAs); it can be the authenticated user
- **Liberty collective security**
  - Involves a discussion of SSL certificates for mutual authentication ... and where those certificates are held
  - Involves a discussion of SSH, SSH keys, or other remote invocation mechanisms, and how Liberty collectives operate

Summary



Application-Related Security

- Encryption
- Authentication
- Authorization

Server-Related Security

- SAF STARTED profiles
- SAF SERVER profiles
- SAF CBIND profiles

File-Related Security

- Product installation files
- Server configuration files

WebSphere Liberty 16.0.0.x Knowledge Center

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