#SHAREorg

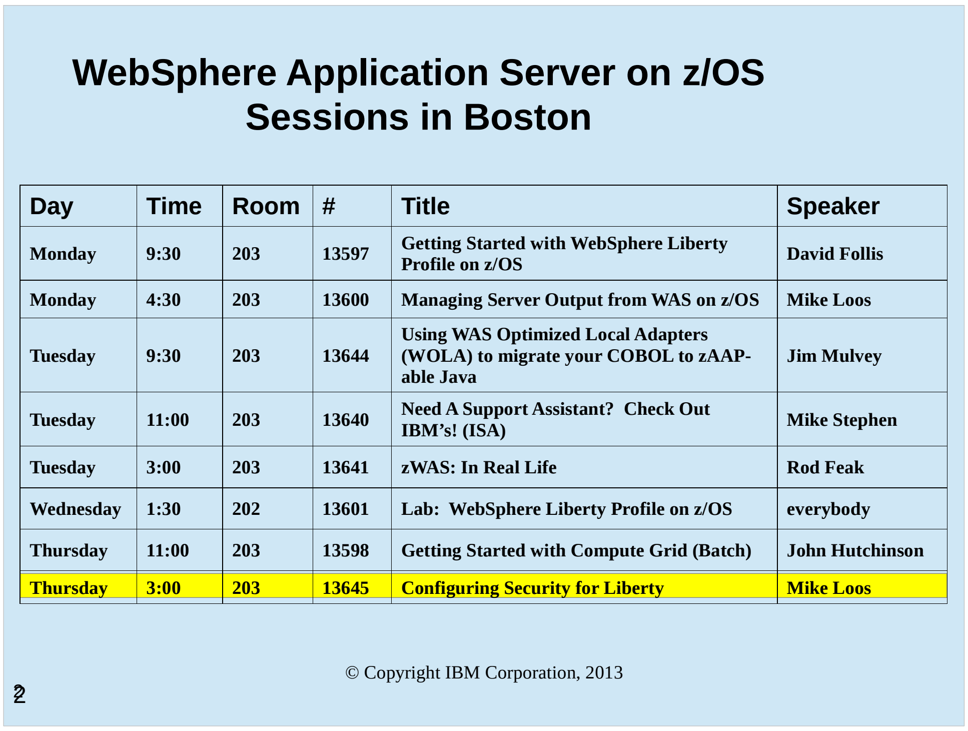
SHARE

Configuring Security for the WebSphere Liberty Profile on z/OS

Mike Loos
IBM
mikeloos@us.ibm.com

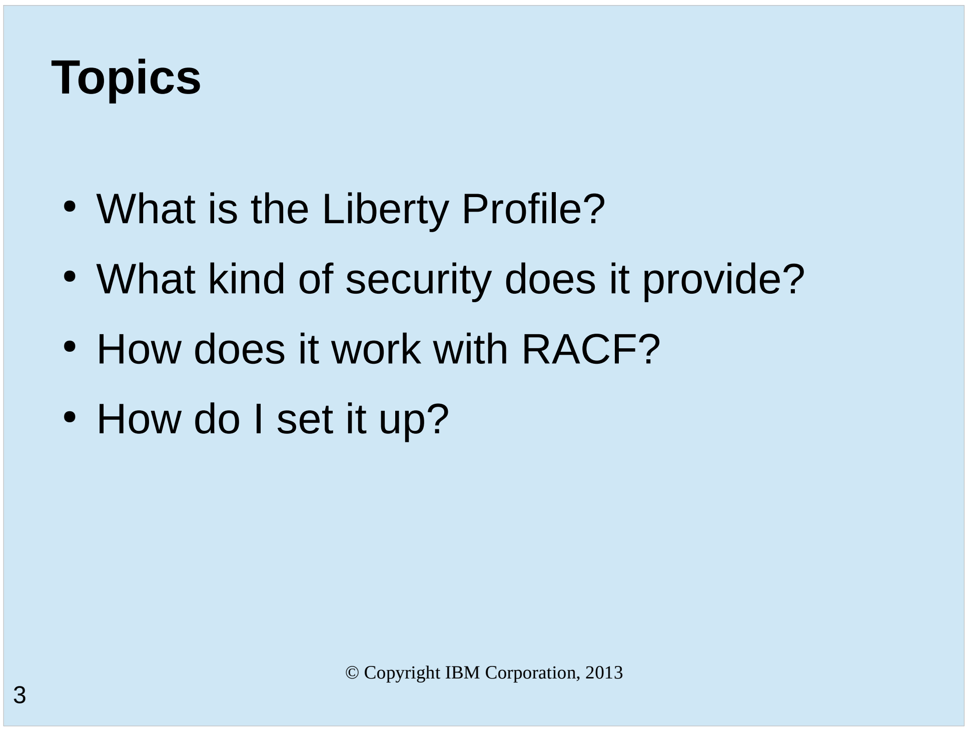
Session number 13645
Thursday, August 15, 2013
3:00 PM

SHARE in Boston



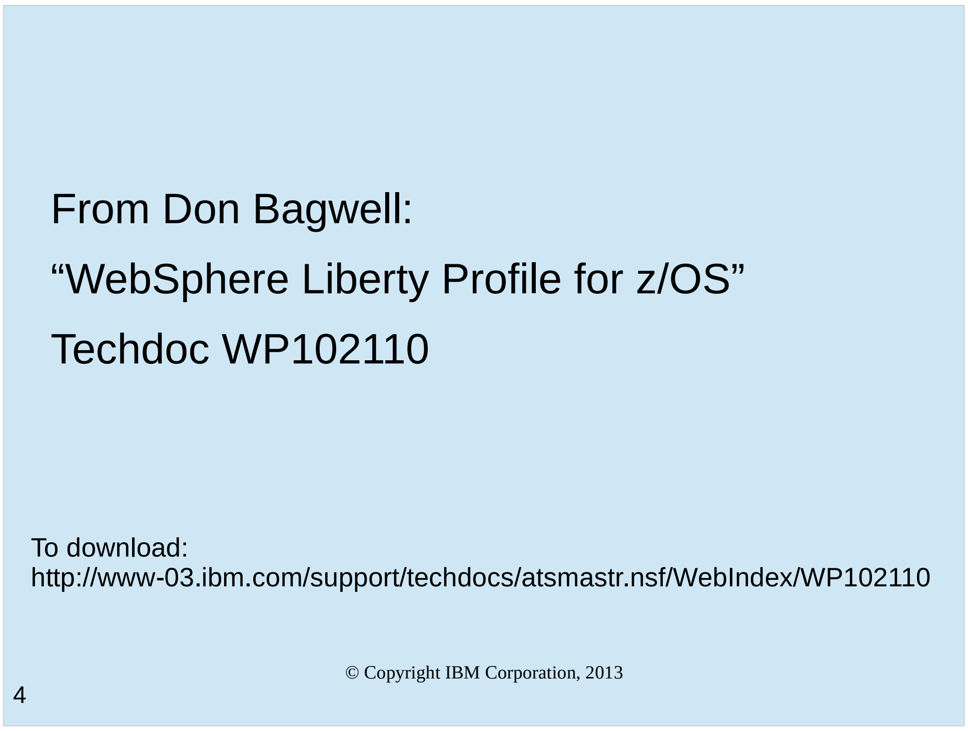
WebSphere Application Server on z/OS Sessions in Boston

Day	Time	Room	#	Title	Speaker
Monday	**9:30**	**203**	**13597**	**Getting Started with WebSphere Liberty Profile on z/OS**	**David Follis**
Monday	**4:30**	**203**	**13600**	**Managing Server Output from WAS on z/OS**	**Mike Loos**
Tuesday	**9:30**	**203**	**13644**	**Using WAS Optimized Local Adapters (WOLA) to migrate your COBOL to zAAP-able Java**	**Jim Mulvey**
Tuesday	**11:00**	**203**	**13640**	**Need A Support Assistant? Check Out IBM's! (ISA)**	**Mike Stephen**
Tuesday	**3:00**	**203**	**13641**	**zWAS: In Real Life**	**Rod Feak**
Wednesday	**1:30**	**202**	**13601**	**Lab: WebSphere Liberty Profile on z/OS**	**everybody**
Thursday	**11:00**	**203**	**13598**	**Getting Started with Compute Grid (Batch)**	**John Hutchinson**
Thursday	**3:00**	**203**	**13645**	**Configuring Security for Liberty**	**Mike Loos**



Topics

- What is the Liberty Profile?
- What kind of security does it provide?
- How does it work with RACF?
- How do I set it up?



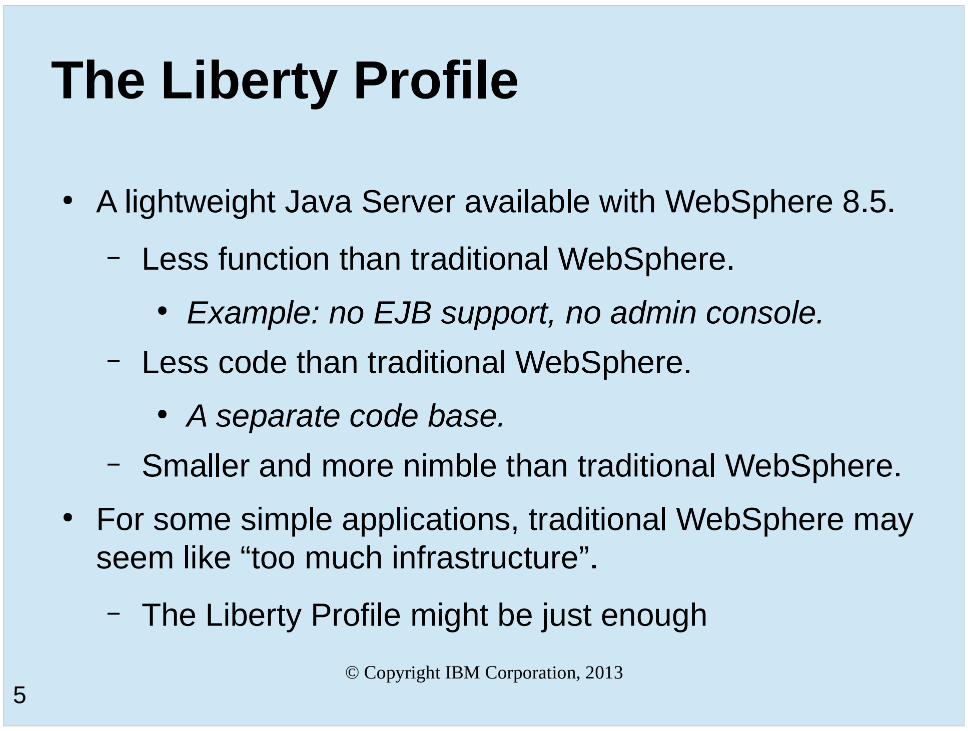
From Don Bagwell:

“WebSphere Liberty Profile for z/OS”

Techdoc WP102110

To download:
http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110

If you're looking for a good document to help you get started with the Liberty profile, I'd recommend this one written by Don Bagwell. This techdoc includes a “quick start” guide that will help you to become familiar with configuring the Liberty profile server.

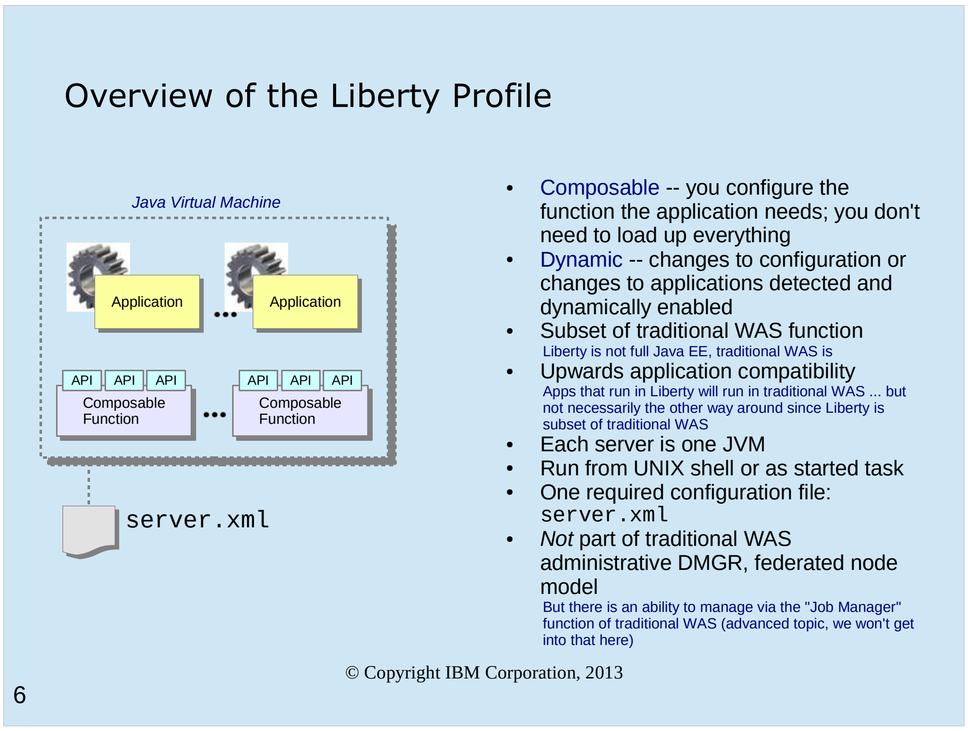


The Liberty Profile

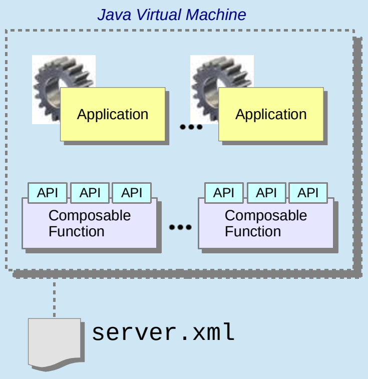
- A lightweight Java Server available with WebSphere 8.5.
  - Less function than traditional WebSphere.
    - *Example: no EJB support, no admin console.*
  - Less code than traditional WebSphere.
    - *A separate code base.*
  - Smaller and more nimble than traditional WebSphere.
- For some simple applications, traditional WebSphere may seem like “too much infrastructure”.
  - The Liberty Profile might be just enough

5

When this foil was originally created, it was accurate. Since the introduction of WebSphere 8.5.5, it is a little out of date. If you are still running WebSphere at the base 8.5 level, through fix pack 2, it is correct. But, as we'll cover on the following slides, the Liberty profile is still a “work in progress”, and more features are being added. The main thing to remember though, is that you only need to configure the features you actually need for your application(s).



Overview of the Liberty Profile



- Composable -- you configure the function the application needs; you don't need to load up everything
- Dynamic -- changes to configuration or changes to applications detected and dynamically enabled
- Subset of traditional WAS function
  Liberty is not full Java EE, traditional WAS is
- Upwards application compatibility
  Apps that run in Liberty will run in traditional WAS ... but not necessarily the other way around since Liberty is subset of traditional WAS
- Each server is one JVM
- Run from UNIX shell or as started task
- One required configuration file: `server.xml`
- *Not* part of traditional WAS administrative DMGR, federated node model
  But there is an ability to manage via the "Job Manager" function of traditional WAS (advanced topic, we won't get into that here)

When approaching the "Liberty Profile" for the first time, you must try to keep it separate from the traditional WAS z/OS server model with the CR, SR and all the things we just finished discussing. The Liberty Profile server model is packaged and delivered with WAS z/OS Version 8.5, but in many ways it is quite different from traditional WAS z/OS.

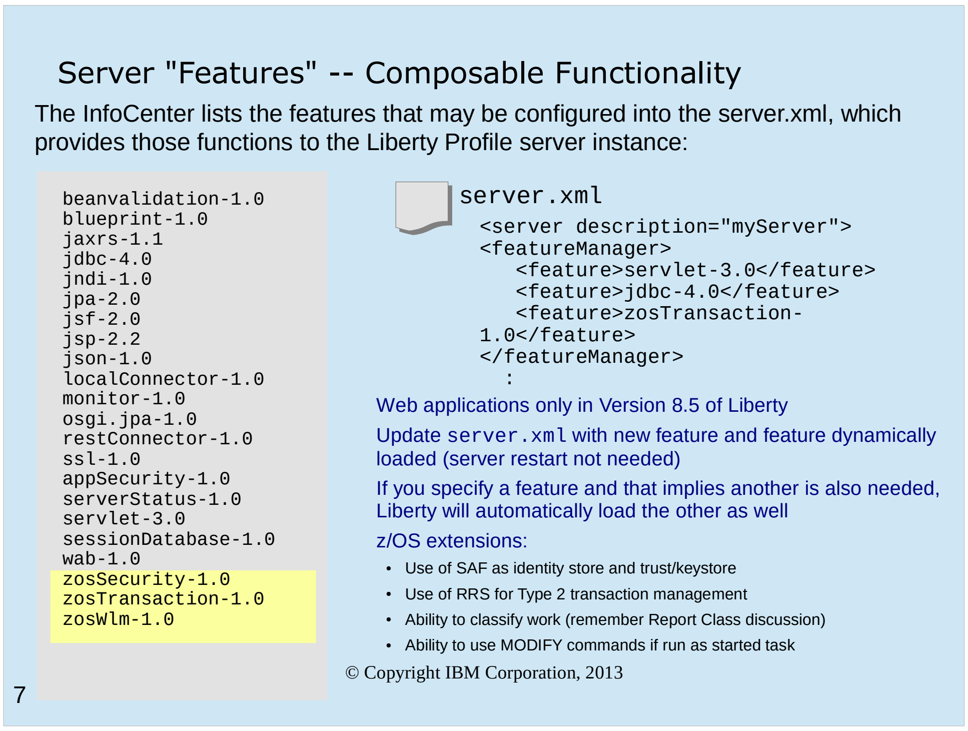
The goal of the Liberty Profile model is to provide a lightweight and flexible server runtime. The lightweight goal is achieved by making the functions loaded by Liberty "composable" -- that is, through the configuration you indicate what functions your application needs and only those functions (and functions Liberty sees are co-requisite with your specified functions) consume resources. The Liberty Profile is also dynamic in that changes to the configuration and changes to the applications may be dynamically detected and updated. (That feature can be controlled so the polling cycle is reduced or turned off completely.)

The Liberty Profile is a *functional subset* of the traditional WAS runtime model. The traditional WAS model is a full Java EE runtime; the Liberty Profile is not. That said, it is important to understand that applications developed and tested with Liberty will run in traditional WAS. (The reverse is not necessarily true ... it's possible to develop an application for the full Java EE WAS runtime that would not work in the functional subset Liberty server.)

Each Liberty Profile server instance is a single JVM, not multiple JVMs like what we saw for traditional WAS z/OS. The server instance may be started and operated from the UNIX shell or as a z/OS started task. Which of those two approaches you select is really a matter of your preference. Running the servers as a z/OS started task provides access to the MODIFY command, but otherwise the function available is the same for both methods.

The configuration is very simple -- one primary XML file ("server.xml") is all it requires. (There are other *optional* configuration files for JVM properties and UNIX environment entries.) Later in this unit we'll discuss what that server.xml file looks like.

The Liberty Profile server instances are not part of the "Admin Console" administration model you may be accustomed to with traditional WAS. Liberty Profile server instances are not part of the traditional WAS z/OS Network Deployment structure. Each server instance is separate from the others but, as we'll see, there is the ability to share configuration and applicaton elements between Liberty servers to aid in scaling up and out.



Server "Features" -- Composable Functionality

The InfoCenter lists the features that may be configured into the server.xml, which provides those functions to the Liberty Profile server instance:

```
beanvalidation-1.0
blueprint-1.0
jaxrs-1.1
jdbc-4.0
jndi-1.0
jpa-2.0
jsf-2.0
jsp-2.2
json-1.0
localConnector-1.0
monitor-1.0
osgi.jpa-1.0
restConnector-1.0
ssl-1.0
appSecurity-1.0
serverStatus-1.0
servlet-3.0
sessionDatabase-1.0
wab-1.0
zosSecurity-1.0
zosTransaction-1.0
zosWlm-1.0
```

server.xml

```
<server description="myServer">
<featureManager>
   <feature>servlet-3.0</feature>
   <feature>jdbc-4.0</feature>
   <feature>zosTransaction-
1.0</feature>
</featureManager>
  :
```

Web applications only in Version 8.5 of Liberty

Update `server.xml` with new feature and feature dynamically loaded (server restart not needed)

If you specify a feature and that implies another is also needed, Liberty will automatically load the other as well

z/OS extensions:

- Use of SAF as identity store and trust/keystore
- Use of RRS for Type 2 transaction management
- Ability to classify work (remember Report Class discussion)
- Ability to use MODIFY commands if run as started task

© Copyright IBM Corporation, 2013

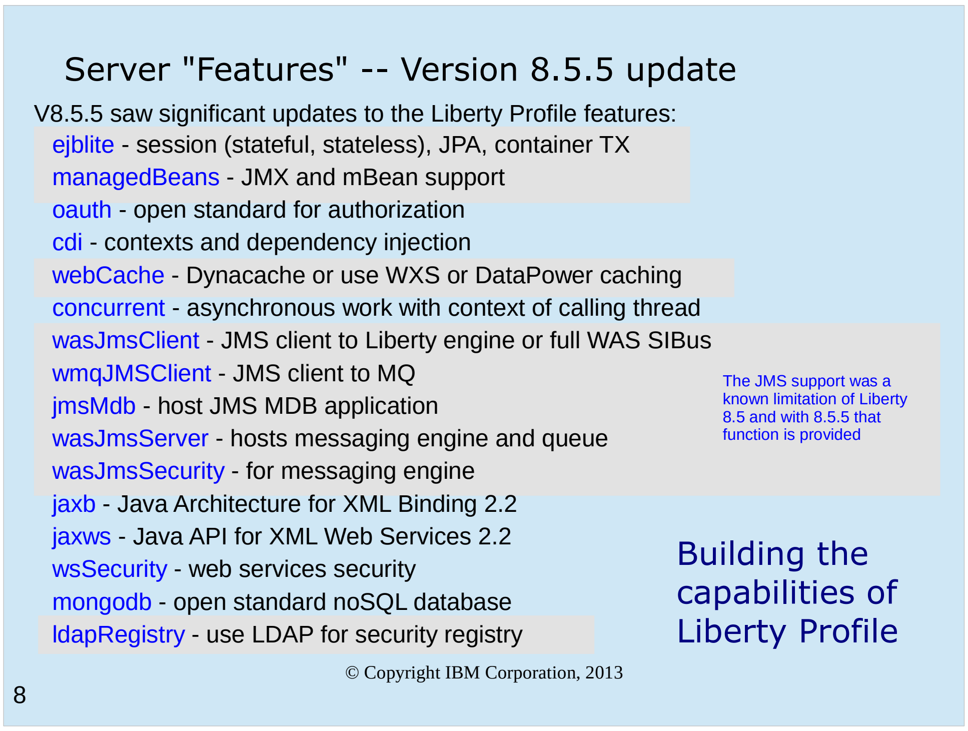
7

On the previous chart's notes we mentioned the "composable" nature of the Liberty Profile server model. In the server.xml file you indicate which features you wish the Liberty Profile server instance to support and load. The chart is showing two things -- all the features supported with the initial release of Liberty (left side of chart) and an example of three of those features being specified in a snippet of server.xml.

Liberty is smart enough to understand pre-requisite and co-requisite functions. So, for example, if you specify "jsp-2.2" Liberty knows it must also load "servlet-3.0" since JSPs become servlets when compiled.

By default Liberty will dynamically monitor the server.xml file for changes, and will update the running server with any changes it detects. Those changes may be additions or deletions to the function.

Liberty Profile for z/OS has a set of extensions designed to take advantage of the z/OS platform. Those extensions are noted on the chart. We'll cover these in more detail later in the unit.



Server "Features" -- Version 8.5.5 update

V8.5.5 saw significant updates to the Liberty Profile features:

- ejblite - session (stateful, stateless), JPA, container TX
- managedBeans - JMX and mBean support
- oauth - open standard for authorization
- cdi - contexts and dependency injection
- webCache - Dynacache or use WXS or DataPower caching
- concurrent - asynchronous work with context of calling thread
- wasJmsClient - JMS client to Liberty engine or full WAS SIBus
- wmqJMSClient - JMS client to MQ
- jmsMdb - host JMS MDB application
- wasJmsServer - hosts messaging engine and queue
- wasJmsSecurity - for messaging engine
- jaxb - Java Architecture for XML Binding 2.2
- jaxws - Java API for XML Web Services 2.2
- wsSecurity - web services security
- mongodb - open standard noSQL database
- ldapRegistry - use LDAP for security registry

The JMS support was a known limitation of Liberty 8.5 and with 8.5.5 that function is provided

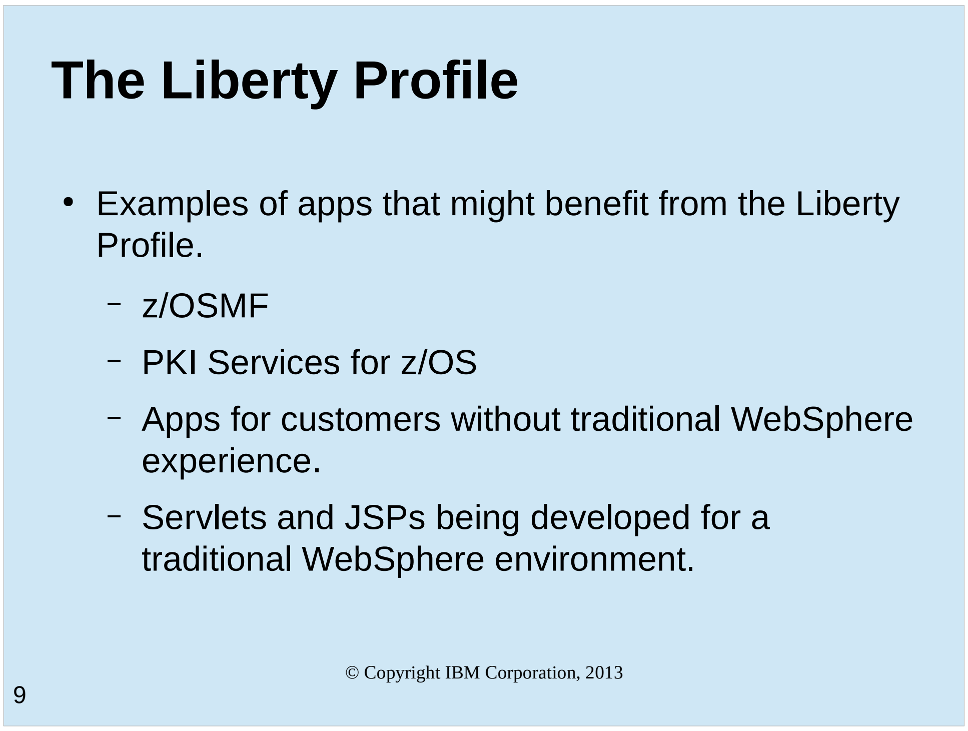
Building the capabilities of Liberty Profile

© Copyright IBM Corporation, 2013

8

When Liberty Profile came out with the V8.5 release, the list of functions available with Liberty was known to be missing a few things. The plan was in place to augment Liberty Profile with those functions over time, and with V8.5.5 we see some of these enhancements.

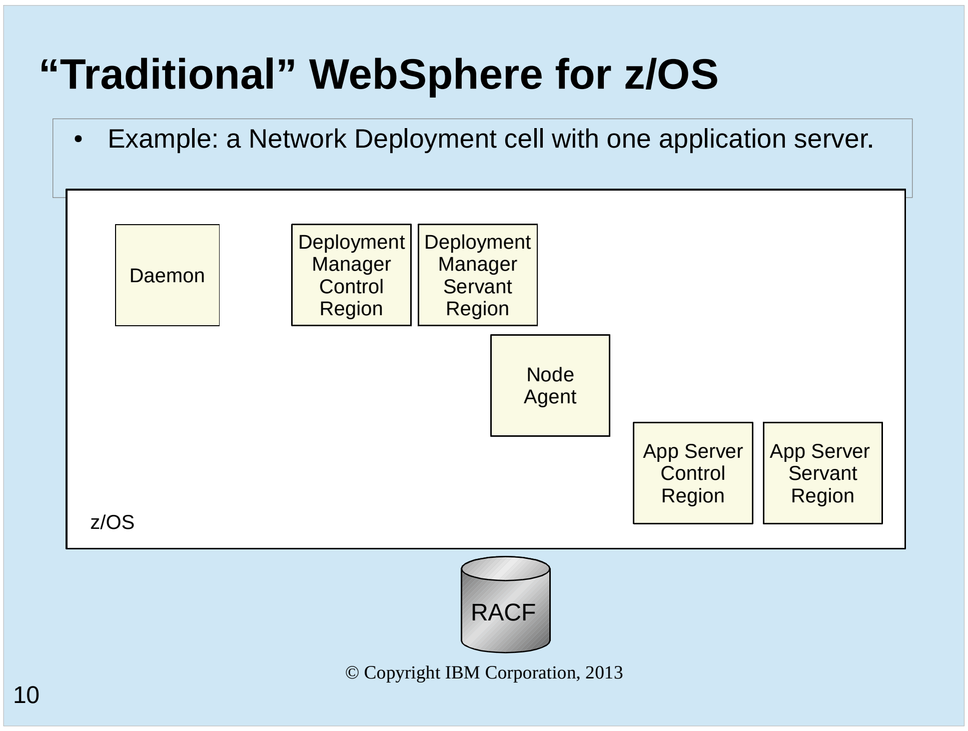
The chart above shows the features added to Liberty Profile V8.5 on z/OS. As you can see, an implementation EJB "lite" comes in, as well as JMS support.



The Liberty Profile

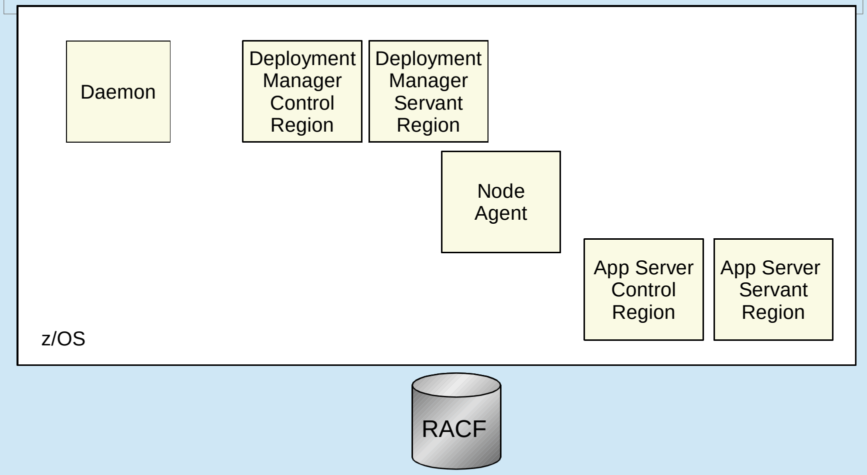
- Examples of apps that might benefit from the Liberty Profile.
  - z/OSMF
  - PKI Services for z/OS
  - Apps for customers without traditional WebSphere experience.
  - Servlets and JSPs being developed for a traditional WebSphere environment.

ZOSMF at the 2.1 level of z/OS uses the Liberty Profile.

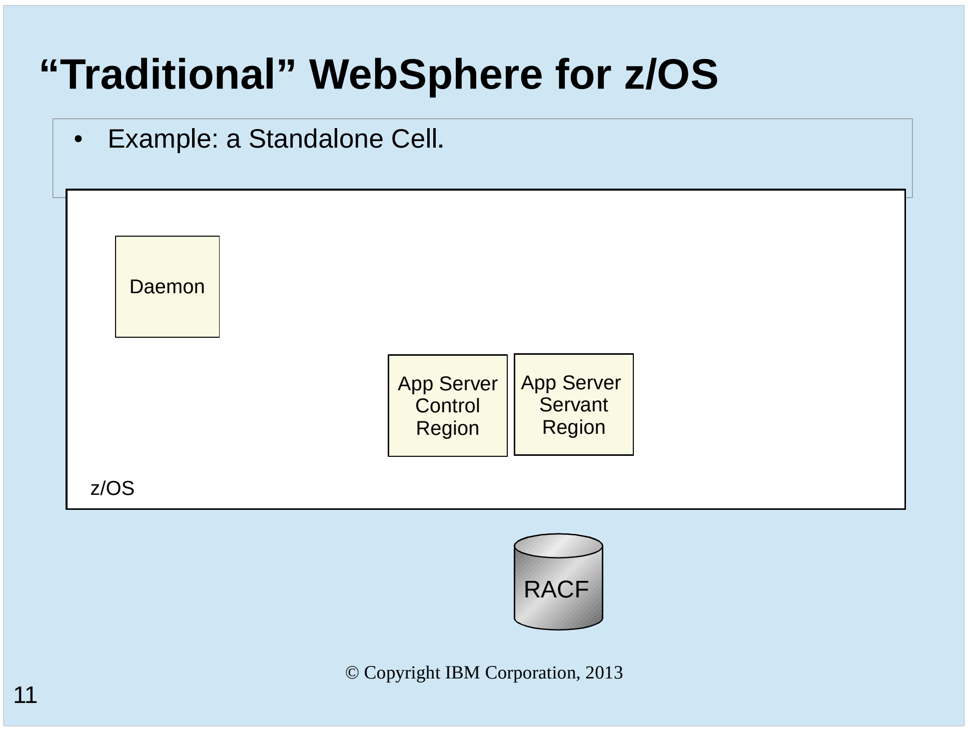


“Traditional” WebSphere for z/OS

- Example: a Network Deployment cell with one application server.


Daemon
Deployment Manager Control Region
Deployment Manager Servant Region
Node Agent
App Server Control Region
App Server Servant Region
z/OS
RACF

This is what a “typical” and minimal traditional WebSphere Application Server on z/OS Network Deployment cell looks like. You have the Daemon address space, and controller and servant for the Deployment Manager server, a nodeagent, and one or more application servers each with a controller and at least one servant.



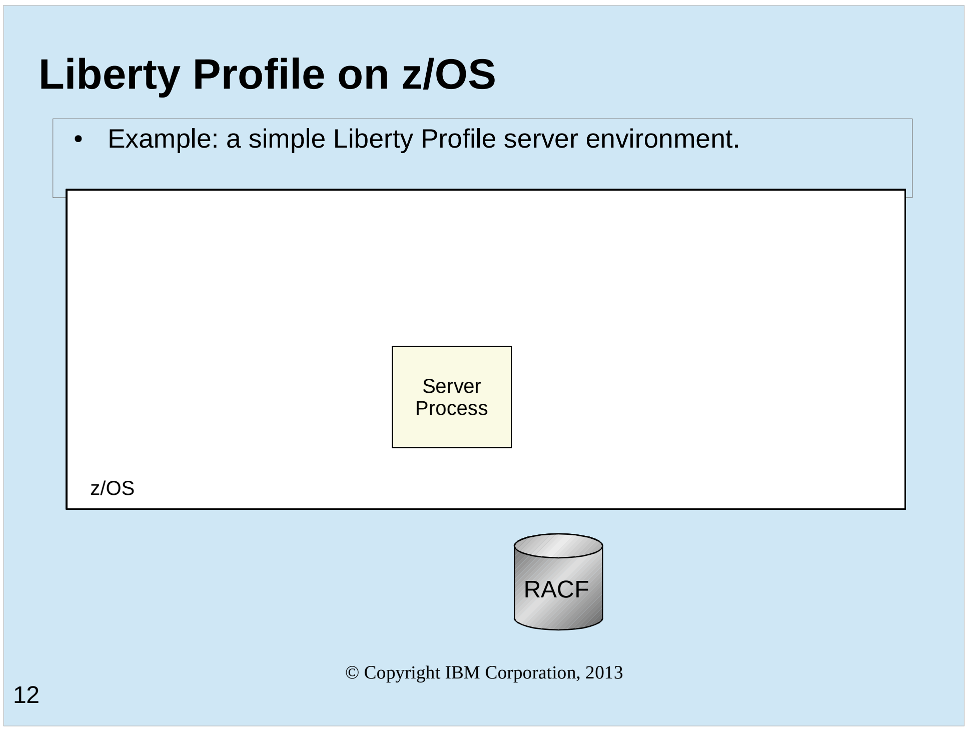
“Traditional” WebSphere for z/OS

- Example: a Standalone Cell.

Daemon
App Server Control Region
App Server Servant Region
z/OS
RACF

11

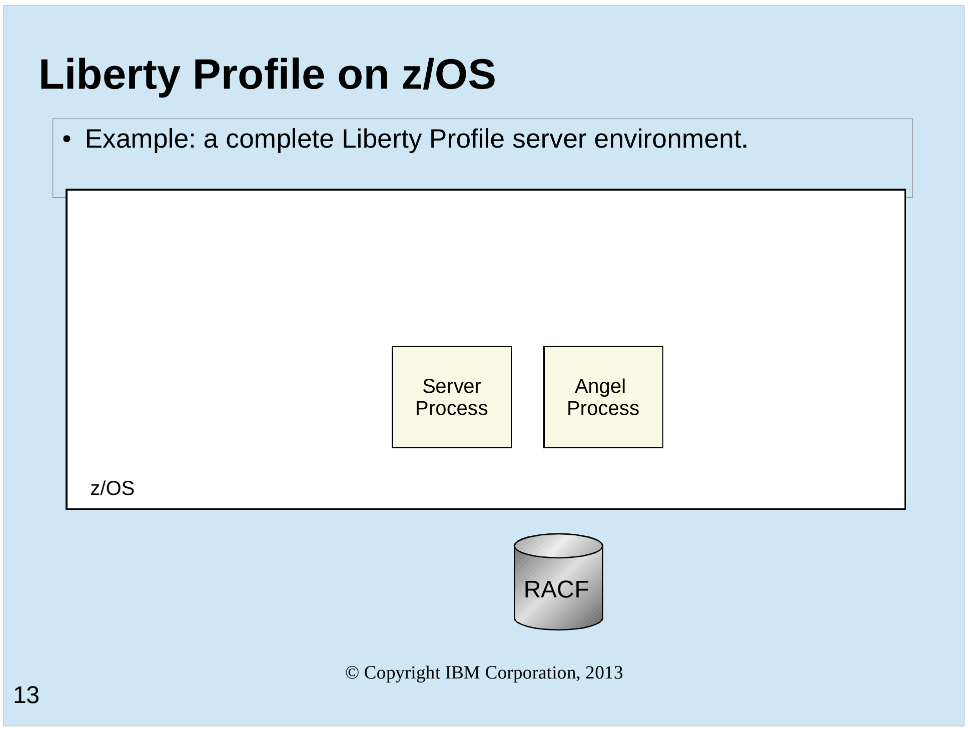
Even if all you have is a Standalone server cell, you still have three address spaces, a Daemon, a controller and a servant.



Liberty Profile on z/OS

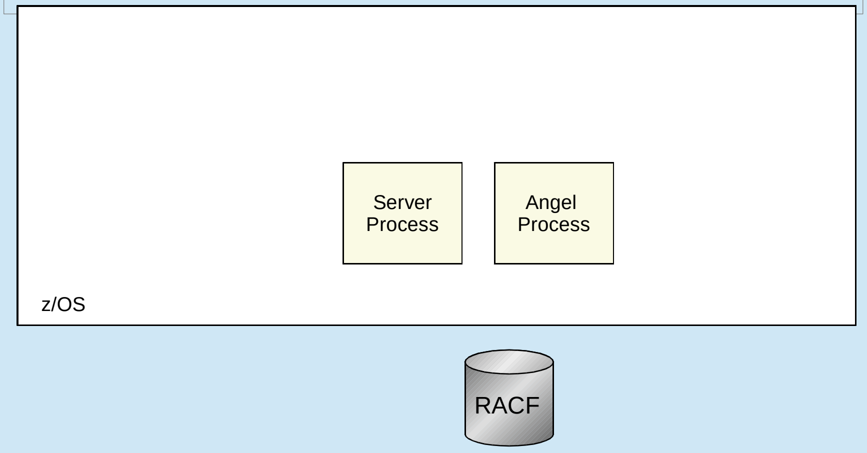
- Example: a simple Liberty Profile server environment.

In contrast to traditional WebSphere on z/OS, the Liberty profile server may be just one address space.



Liberty Profile on z/OS

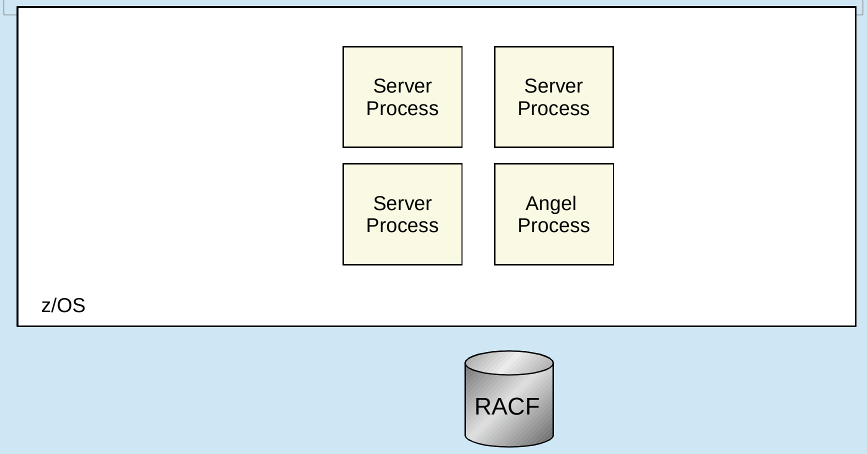
- Example: a complete Liberty Profile server environment.



A complete environment for a single server consists of only two address spaces, the “Angel” process address space, and the actual server process address space.

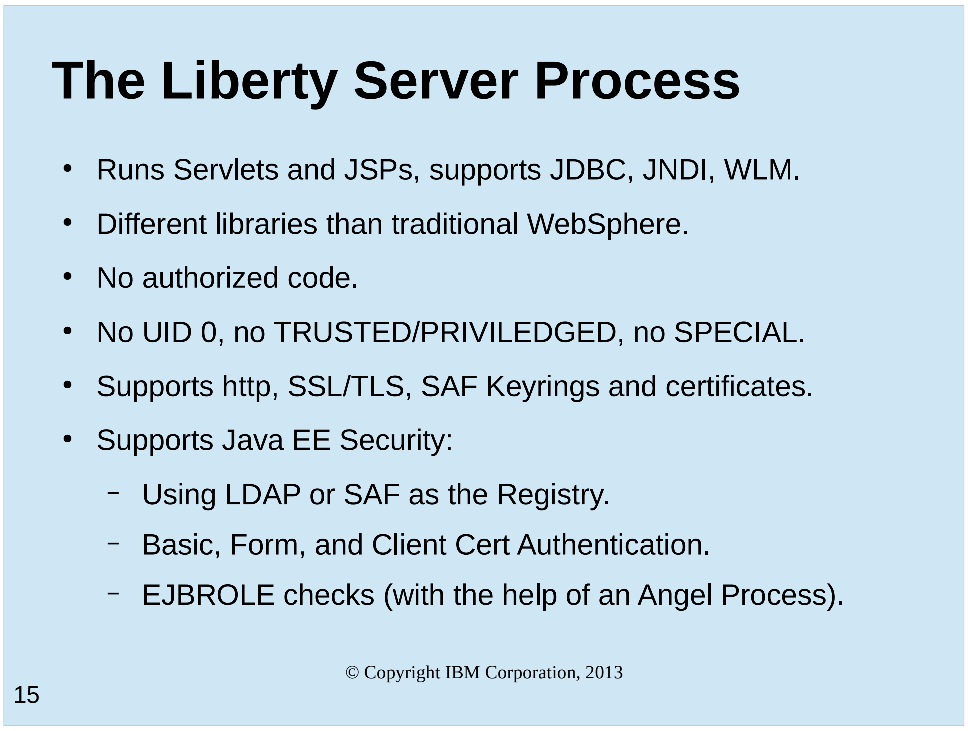
Liberty Profile on z/OS

- Example: multiple (3) complete Liberty Profile server environments.



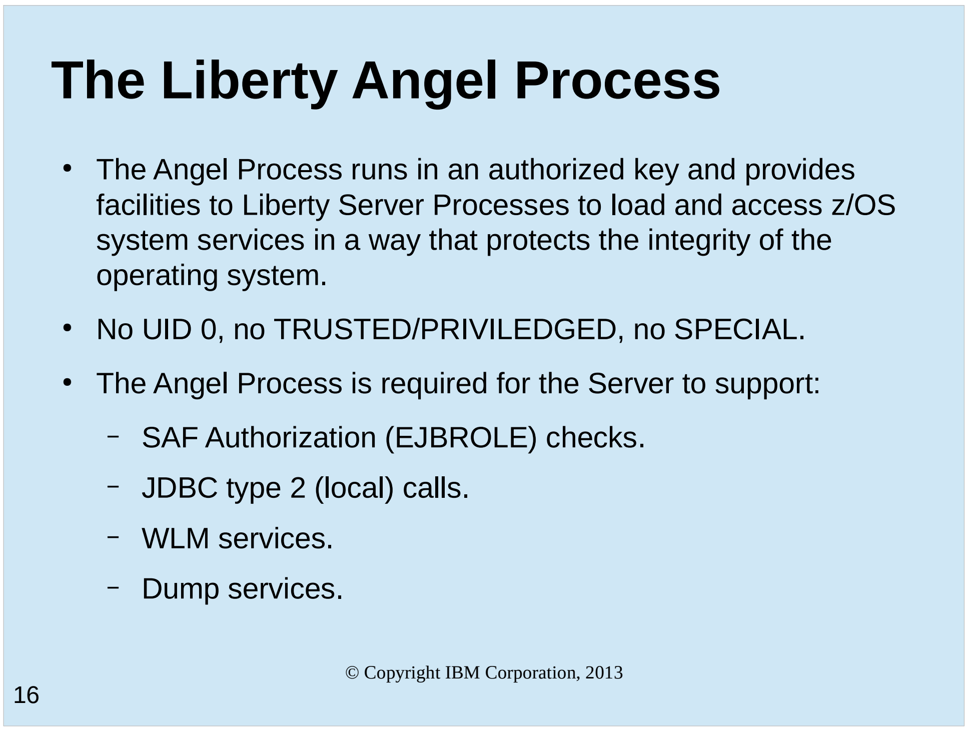
14

Multiple Liberty profile server process address spaces can all use the same angel process.



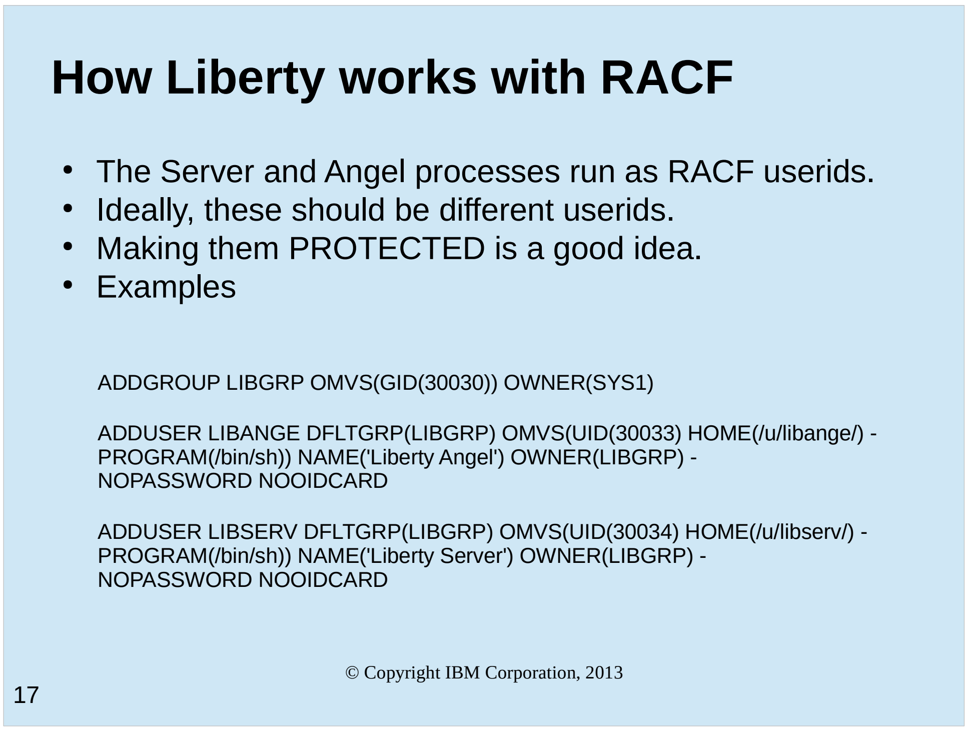
The Liberty Server Process

- Runs Servlets and JSPs, supports JDBC, JNDI, WLM.
- Different libraries than traditional WebSphere.
- No authorized code.
- No UID 0, no TRUSTED/PRIVILEDGED, no SPECIAL.
- Supports http, SSL/TLS, SAF Keyrings and certificates.
- Supports Java EE Security:
  - Using LDAP or SAF as the Registry.
  - Basic, Form, and Client Cert Authentication.
  - EJBROLE checks (with the help of an Angel Process).



The Liberty Angel Process

- The Angel Process runs in an authorized key and provides facilities to Liberty Server Processes to load and access z/OS system services in a way that protects the integrity of the operating system.
- No UID 0, no TRUSTED/PRIVILEDGED, no SPECIAL.
- The Angel Process is required for the Server to support:
  - SAF Authorization (EJBROLE) checks.
  - JDBC type 2 (local) calls.
  - WLM services.
  - Dump services.



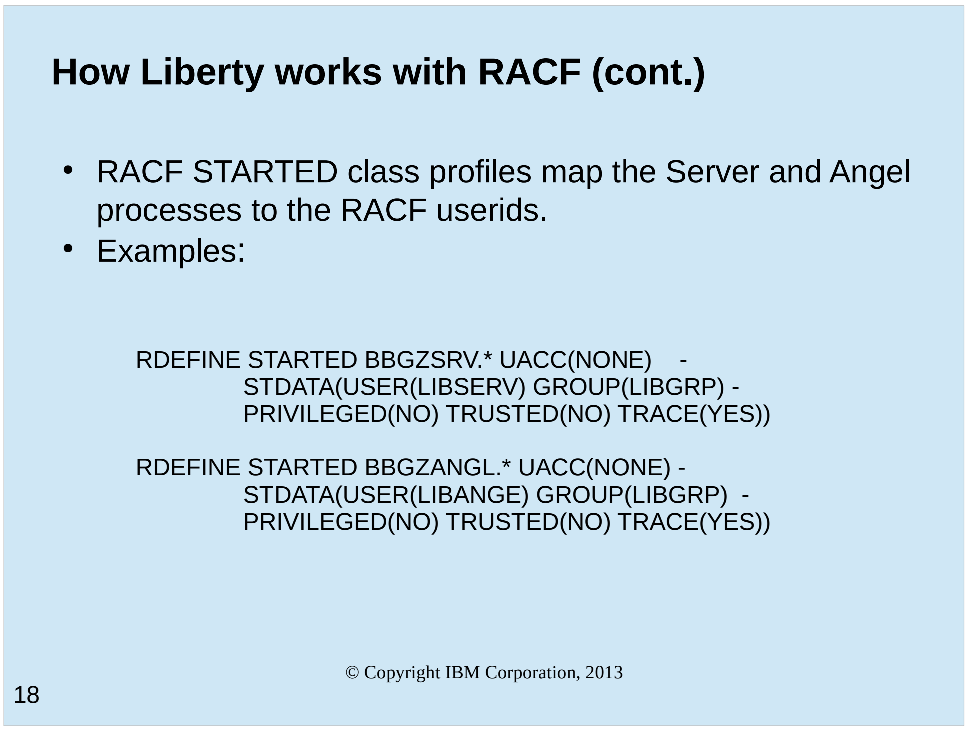
How Liberty works with RACF

- The Server and Angel processes run as RACF userids.
- Ideally, these should be different userids.
- Making them PROTECTED is a good idea.
- Examples

```
ADDGROUP LIBGRP OMVS(GID(30030)) OWNER(SYS1)

ADDUSER LIBANGE DFLTGRP(LIBGRP) OMVS(UID(30033) HOME(/u/libange/) -
PROGRAM(/bin/sh)) NAME('Liberty Angel') OWNER(LIBGRP) -
NOPASSWORD NOOIDCARD

ADDUSER LIBSERV DFLTGRP(LIBGRP) OMVS(UID(30034) HOME(/u/libserv/) -
PROGRAM(/bin/sh)) NAME('Liberty Server') OWNER(LIBGRP) -
NOPASSWORD NOOIDCARD
```

While the Liberty profile server can run as a process started from a shell (OMVS, telnet, ssh), it is expected that any production servers would be more likely to run as a started task. Running the server as a started task allows you to interact with the server via the z/OS modify command as well as the stop command.

Security setup to run the Liberty profile server and angel requires that you add a group, and one or more userids. Making these userids “Protected”, prevents someone else from logging on as that userid or running a batch job as that userid (unless they have surrogat authority to it).

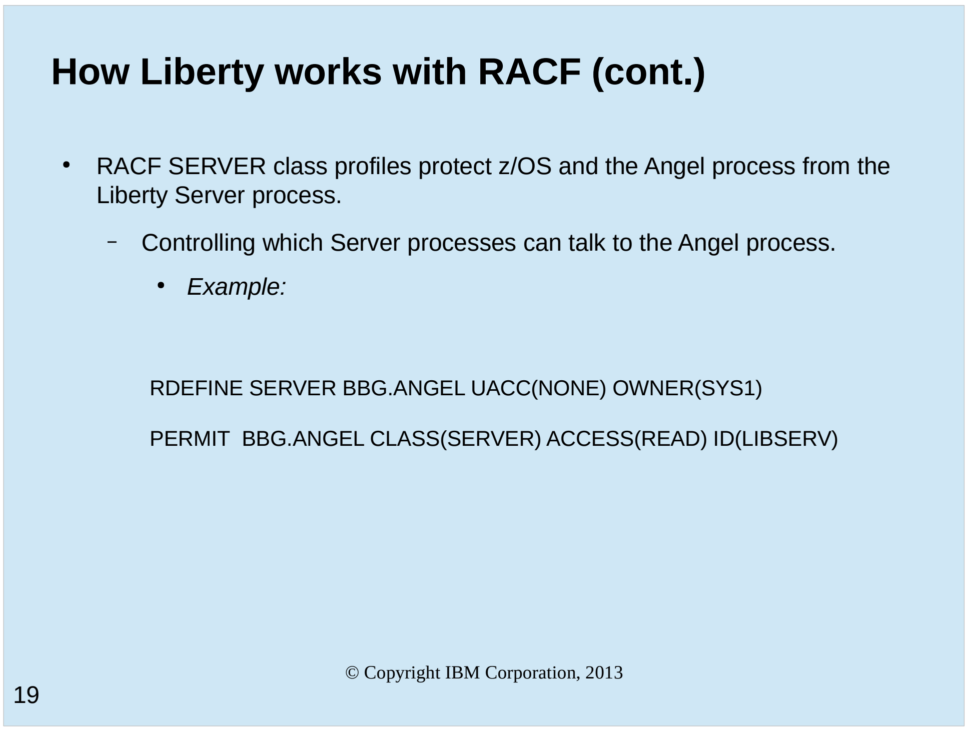


How Liberty works with RACF (cont.)

- RACF STARTED class profiles map the Server and Angel processes to the RACF userids.
- Examples:

```
RDEFINE STARTED BBGZSRV.* UACC(NONE)    -
          STDATA(USER(LIBSERV) GROUP(LIBGRP) -
          PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))

RDEFINE STARTED BBGZANGL.* UACC(NONE) -
          STDATA(USER(LIBANGE) GROUP(LIBGRP)  -
          PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

You'll also want STARTED class profiles to allow the system to assign the proper userid and group to the started task address space.



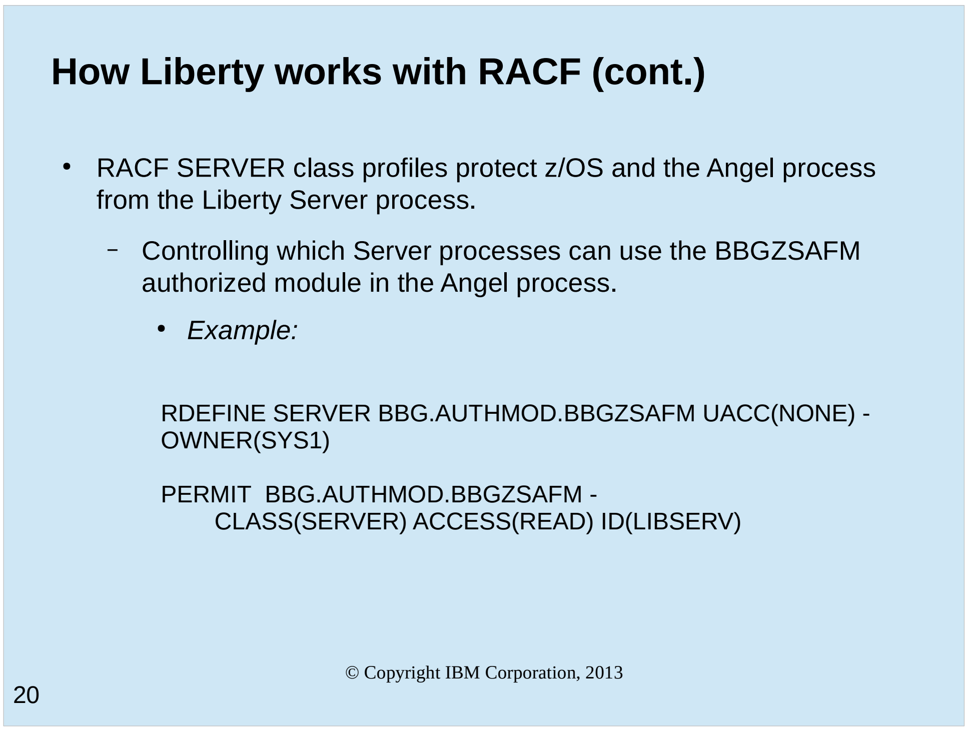
How Liberty works with RACF (cont.)

- RACF SERVER class profiles protect z/OS and the Angel process from the Liberty Server process.
  - Controlling which Server processes can talk to the Angel process.
    - *Example:*

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)

PERMIT  BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

19

WebSphere on z/OS uses SERVER class profiles to determine which other processes may access its services. The SERVER profile we see here is for the angel address space and we are allowing (with the RACF PERMIT command), the id running the server process (LIBSERV) to access the angel processes services. This is one reason why you may want to run each server (or group of servers) with unique userids. You may not want all servers to have access to the angel process's services.

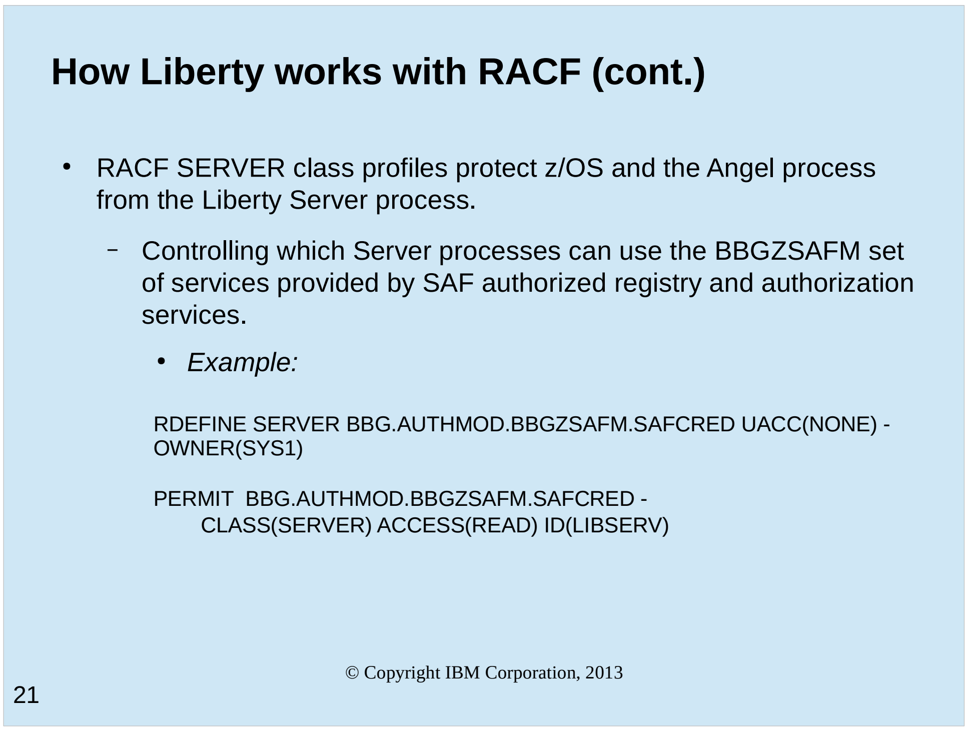


How Liberty works with RACF (cont.)

- RACF SERVER class profiles protect z/OS and the Angel process from the Liberty Server process.
  - Controlling which Server processes can use the BBGZSAFM authorized module in the Angel process.
    - *Example:*

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) -
OWNER(SYS1)

PERMIT  BBG.AUTHMOD.BBGZSAFM -
        CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

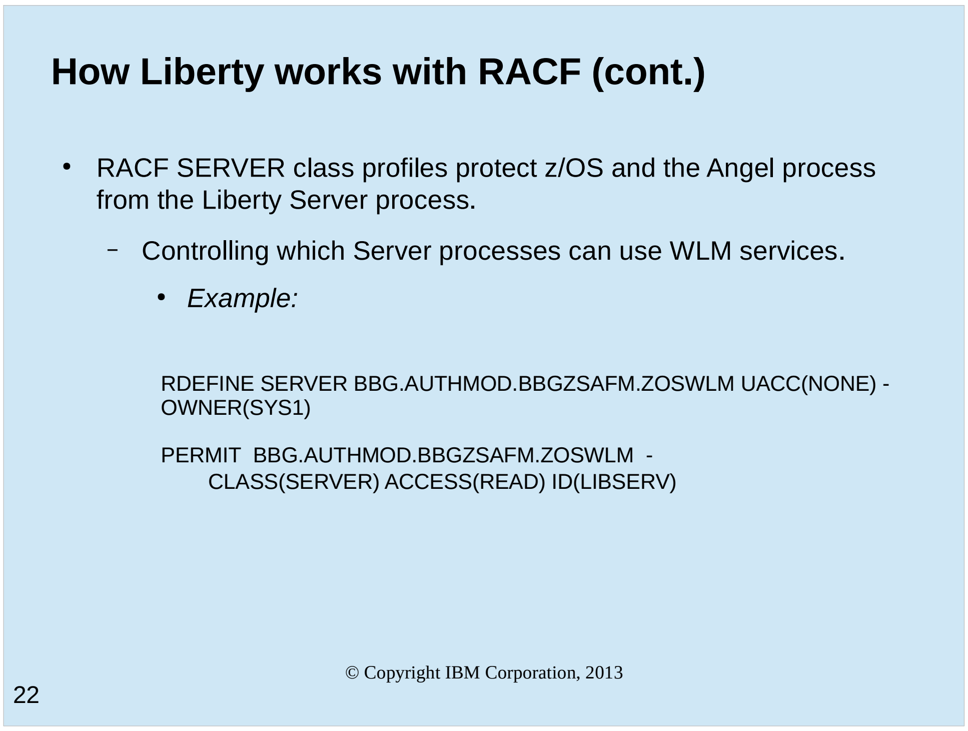
These commands create the SERVER profile is for the authorized module BBGZSAFM and permits the Liberty server Started Task user ID to the profile. This action allows a Liberty server to use the z/OS Authorized services The PERMIT allows a server running as LIBSERV to access the authorized module.


How Liberty works with RACF (cont.)

- RACF SERVER class profiles protect z/OS and the Angel process from the Liberty Server process.
  - Controlling which Server processes can use the BBGZSAFM set of services provided by SAF authorized registry and authorization services.
    - *Example:*

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE) -
OWNER(SYS1)

PERMIT  BBG.AUTHMOD.BBGZSAFM.SAFCRED -
    CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

This profile protects the use of the SAF authorized user registry services and SAF authorization services (SAFCRED).



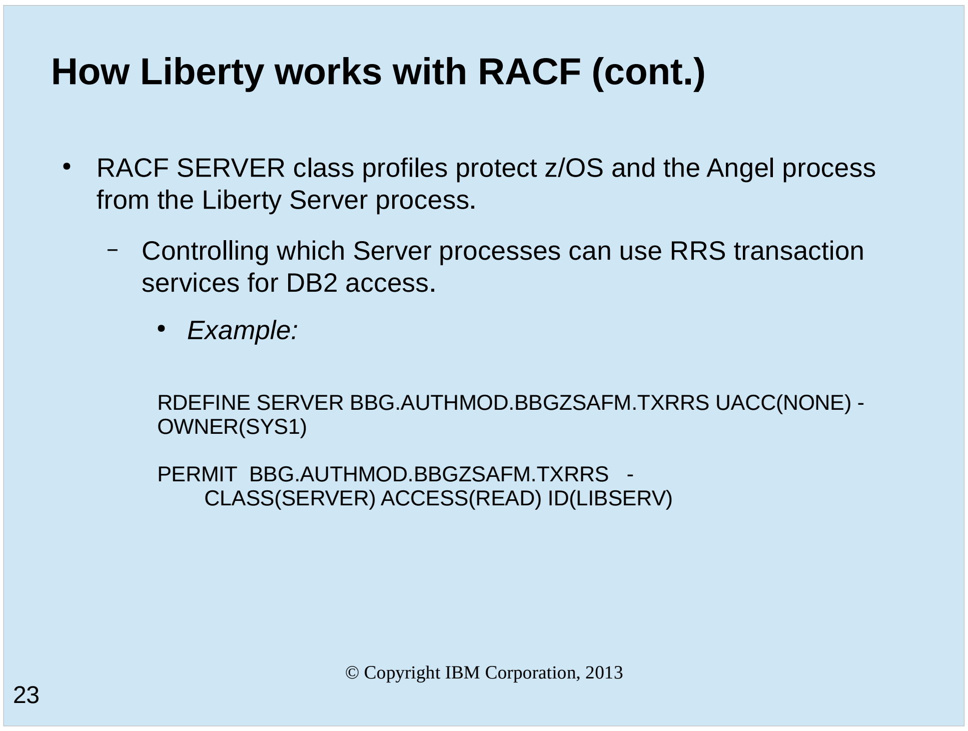
How Liberty works with RACF (cont.)

- RACF SERVER class profiles protect z/OS and the Angel process from the Liberty Server process.
  - Controlling which Server processes can use WLM services.
    - *Example:*

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) -
OWNER(SYS1)

PERMIT  BBG.AUTHMOD.BBGZSAFM.ZOSWLM  -
       CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

22

This profile protects the use of the authorized WLM services (ZOSWLM). If the server doesn't have this access it will use the language environment unauthorized versions of these services for interaction with z/OS Workload Manager.

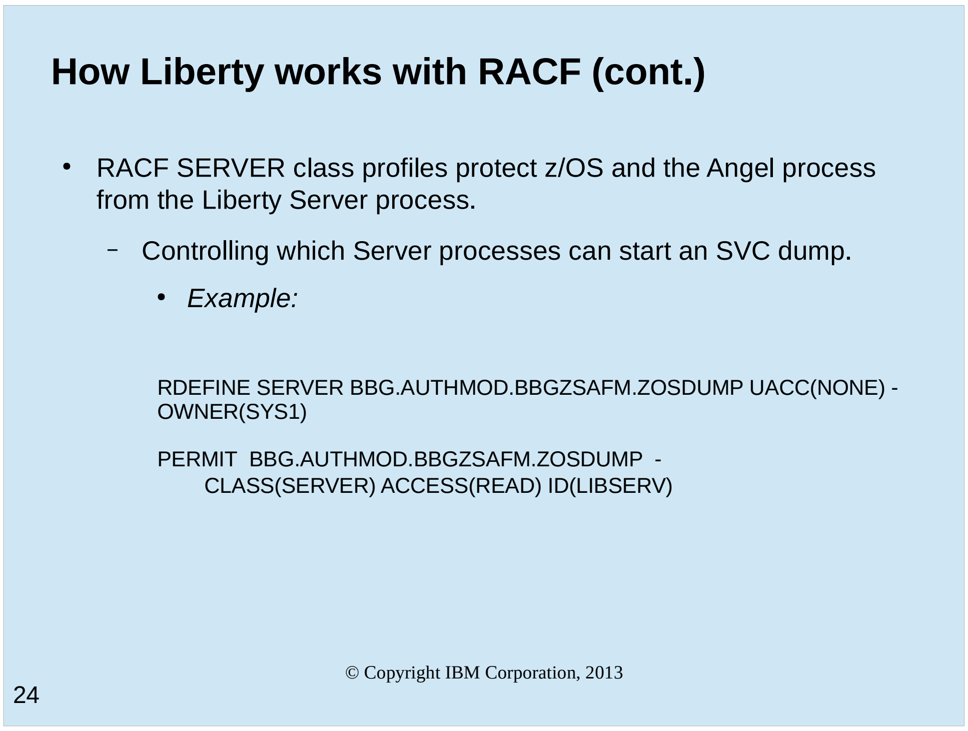


How Liberty works with RACF (cont.)

- RACF SERVER class profiles protect z/OS and the Angel process from the Liberty Server process.
  - Controlling which Server processes can use RRS transaction services for DB2 access.
    - *Example:*

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) -
OWNER(SYS1)

PERMIT  BBG.AUTHMOD.BBGZSAFM.TXRRS   -
      CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

This profile protects the use of RRS transaction services (TXRRS).



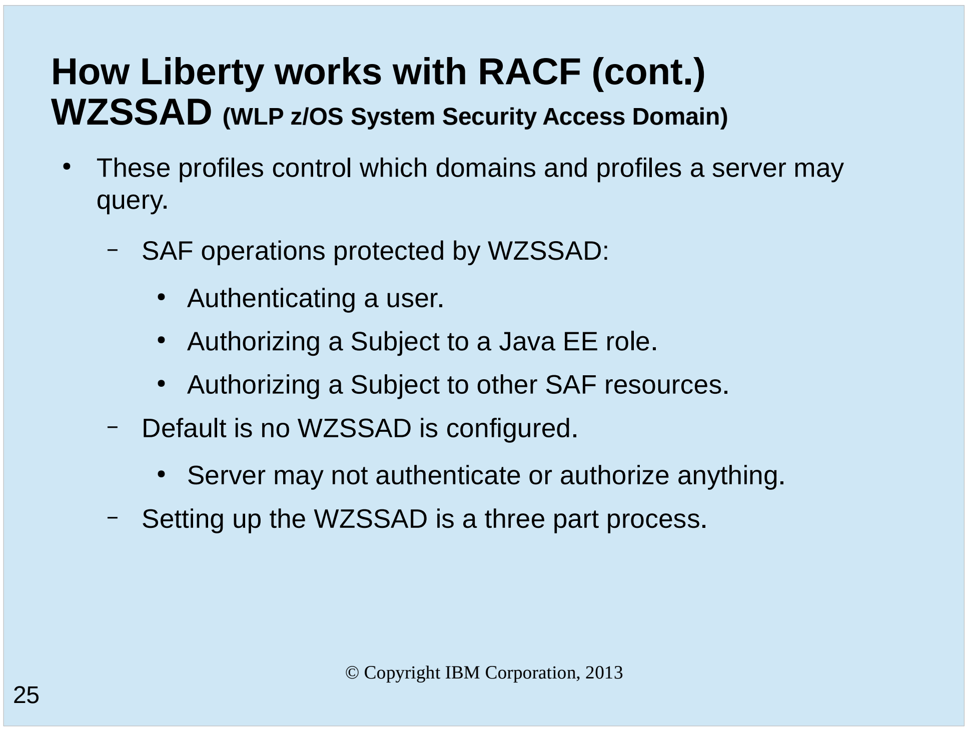
How Liberty works with RACF (cont.)

- RACF SERVER class profiles protect z/OS and the Angel process from the Liberty Server process.
  - Controlling which Server processes can start an SVC dump.
    - *Example:*

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) -
OWNER(SYS1)

PERMIT  BBG.AUTHMOD.BBGZSAFM.ZOSDUMP  -
       CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

24

This profile protects the use of SVCDUMP services (ZOSDUMP).



How Liberty works with RACF (cont.)
WZSSAD (WLP z/OS System Security Access Domain)

- These profiles control which domains and profiles a server may query.
  - SAF operations protected by WZSSAD:
    - Authenticating a user.
    - Authorizing a Subject to a Java EE role.
    - Authorizing a Subject to other SAF resources.
  - Default is no WZSSAD is configured.
    - Server may not authenticate or authorize anything.
  - Setting up the WZSSAD is a three part process.

The below text is reproduced from the WebSphere 8.5 InfoCenter.

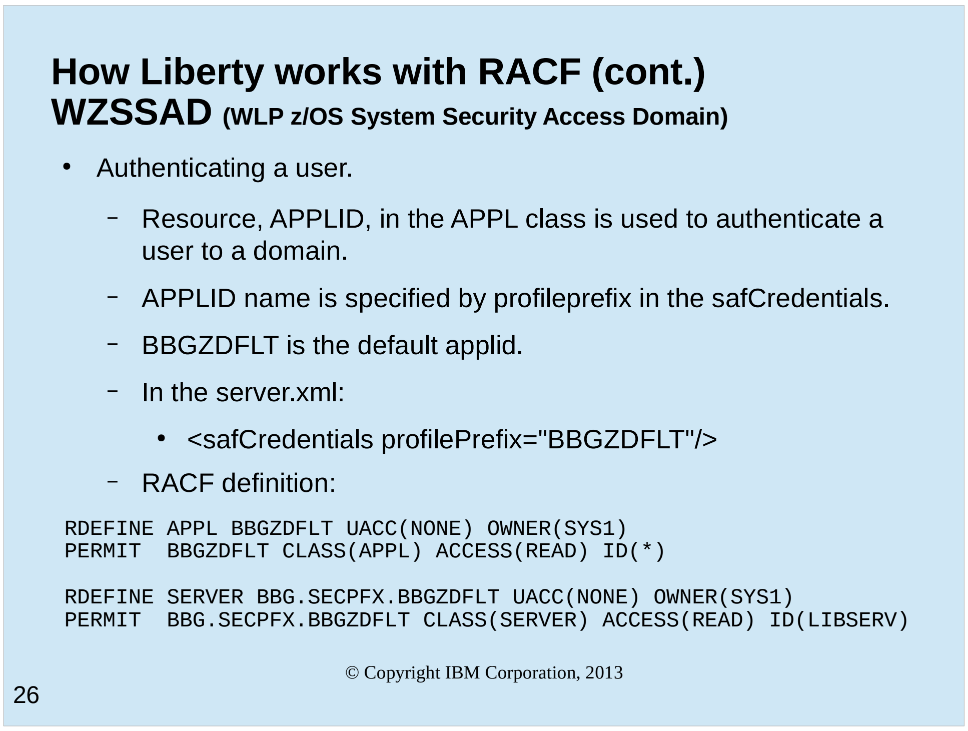
The WLP z/OS® System Security Access Domain (WZSSAD) refers to the permissions granted to the Liberty profile server. These permissions control which System Authorization Facility (SAF) application domains and resource profiles the server is permitted to query when authenticating and authorizing users.

For example, if you want to set up two Liberty server instances, one for production and one for test and you want your role access to be different between production and test, you can set up two different System Security Access Domains.

The Liberty server is an unauthorized program that can be configured and run by non-administrator or non-privileged users, so it is important for system security and integrity purposes that the user is prevented from leveraging the server to run security operations for which they have not been explicitly granted permission.
Note: The WZSSAD is in effect only when the server is using authorized SAF services for authentication and authorization. This means the angel process is running, and the server has been granted permission to use the SAFCRED authorized service routines.
There are three SAF operations that are protected by the WZSSAD:

Authenticating a user
Authorizing a Subject to a Java EE role
Authorizing a Subject to other SAF resources

By default, the WZSSAD is not configured, which means the server has no permission to authenticate or authorize anything. For example, the Liberty server cannot use authorized SAF services for authentication or authorization until an administrator grants it permission to perform some or all of the operations.



How Liberty works with RACF (cont.)
WZSSAD (WLP z/OS System Security Access Domain)

- Authenticating a user.
  - Resource, APPLID, in the APPL class is used to authenticate a user to a domain.
  - APPLID name is specified by profileprefix in the safCredentials.
  - BBGZDFLT is the default applid.
  - In the server.xml:
    - <safCredentials profilePrefix="BBGZDFLT"/>
  - RACF definition:

```
RDEFINE APPL BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT  BBGZDFLT CLASS(APPL) ACCESS(READ) ID(*)

RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT  BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

26

The below text is reproduced from the WebSphere 8.5 InfoCenter.

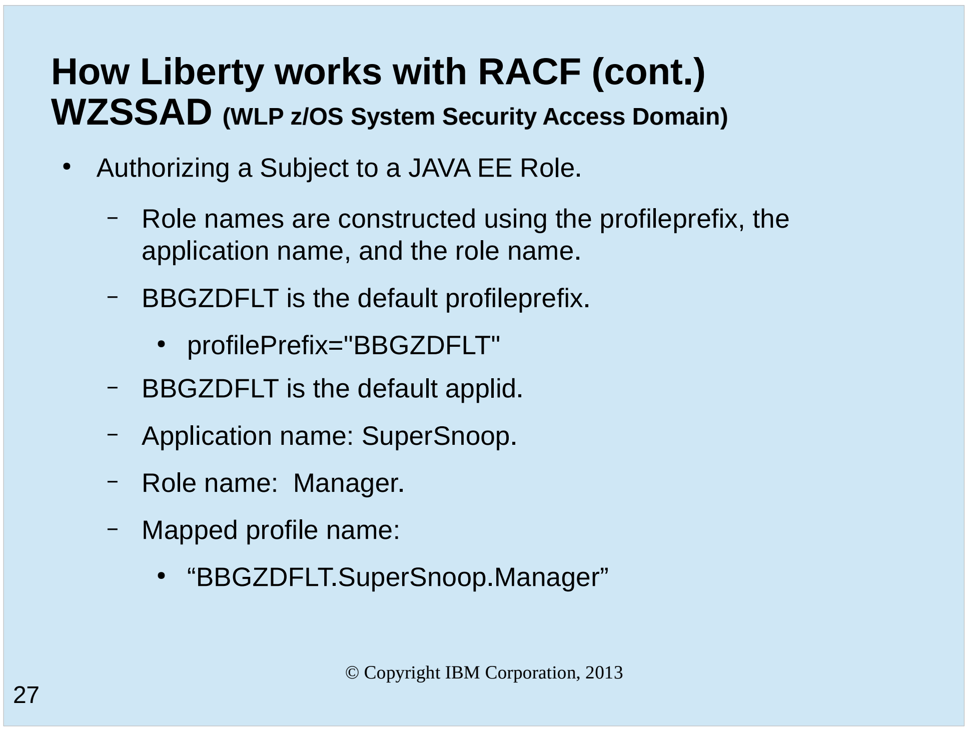
The server authenticates users to a particular SAF domain that is configured by defining a resource, which is referred to as the APPLID in the APPL class. In order to authenticate a user to the domain, the user must have READ access to the APPLID resource in the APPL class.

The APPLID resource name used by the server is specified by the profilePrefix attribute in the <safCredentials> configuration element. If you do not specify this element, then the default profilePrefix of BBGZDFLT is used.

If not active, the domain is not restricted, which means anyone can authenticate to it.

All users to be authenticated by the server must have READ access to the APPLID in the APPL class.

In addition, the server must be granted permission within the WZSSAD to make authentication calls in the given APPLID domain. This prevents an unauthorized user from leveraging the server's use of authorized SAF services to discover information about which APPLIDs can and cannot be authenticated. To grant the server permission to authenticate in a particular APPLID domain, the started task ID of the Liberty server must be granted READ access to the BBG.SECPFX.<APPLID> profile in the SERVER class



How Liberty works with RACF (cont.) WZSSAD (WLP z/OS System Security Access Domain)

- Authorizing a Subject to a JAVA EE Role.
  - Role names are constructed using the profileprefix, the application name, and the role name.
  - BBGZDFLT is the default profileprefix.
    - profilePrefix="BBGZDFLT"
  - BBGZDFLT is the default applid.
  - Application name: SuperSnoop.
  - Role name: Manager.
  - Mapped profile name:
    - “BBGZDFLT.SuperSnoop.Manager”

27

The below text is reproduced from the WebSphere 8.5 InfoCenter.

Authorizing a Subject to a Java EE role

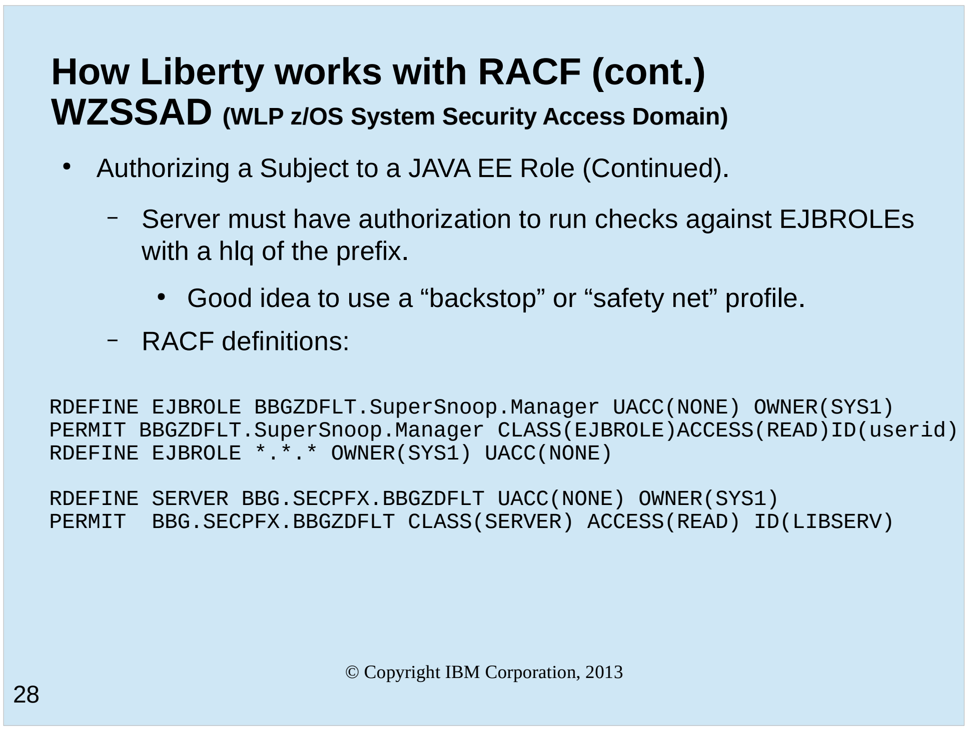
The server authorizes a subject against a Java EE application security role name by checking whether the subject is authorized to a SAF resource profile defined in the EJBROLE class. The SAF resource profile name is mapped from the application role name via the SAF role mapper. The SAF role mapper generates a SAF profile name for a given application role name and application resource name. By default, it generates the SAF profile name using the pattern {profilePrefix}.{resource}.{role}. For example:

profilePrefix="BBGZDFLT"

Application resource name = "SuperSnoop"

Application role name = "Manager"

Mapped profile name = "BBGZDFLT.SuperSnoop.Manager"



How Liberty works with RACF (cont.)
WZSSAD **(WLP z/OS System Security Access Domain)**

- Authorizing a Subject to a JAVA EE Role (Continued).
  - Server must have authorization to run checks against EJBROLEs with a hlq of the prefix.
    - Good idea to use a “backstop” or “safety net” profile.
  - RACF definitions:

```
RDEFINE EJBROLE BBGZDFLT.SuperSnoop.Manager UACC(NONE) OWNER(SYS1)
PERMIT BBGZDFLT.SuperSnoop.Manager CLASS(EJBROLE)ACCESS(READ)ID(userid)
RDEFINE EJBROLE *.*.* OWNER(SYS1) UACC(NONE)

RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT  BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

28

The below text is reproduced from the WebSphere 8.5 InfoCenter.

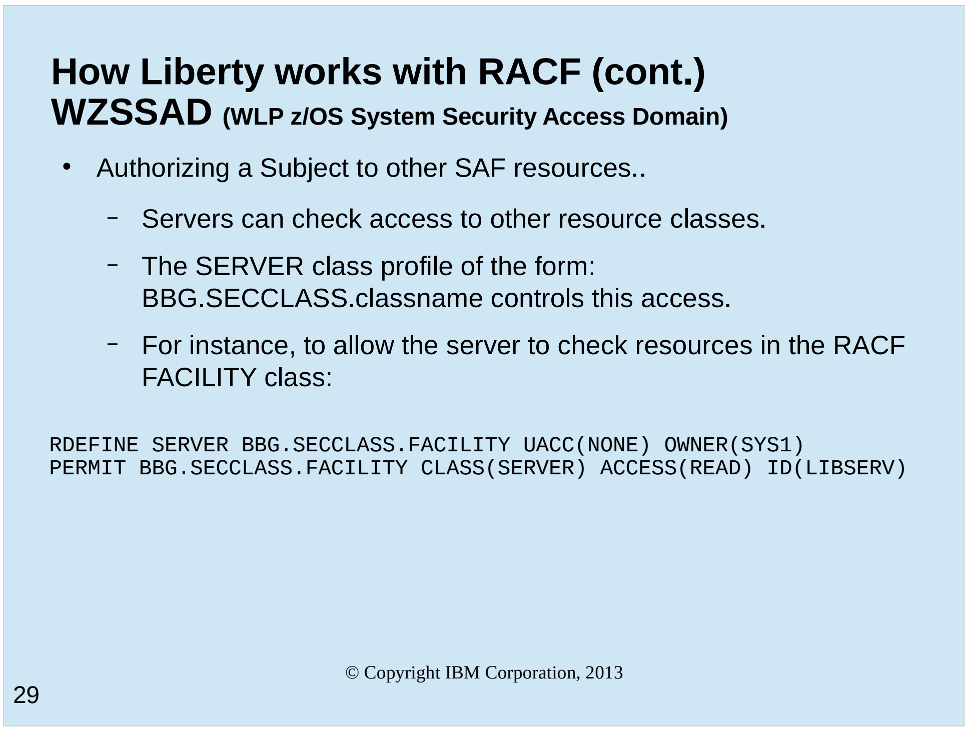
The WZSSAD restricts which SAF profiles in the EJBROLE class the server is allowed to perform authorization against. This prevents an unauthorized user from leveraging the server's use of authorized SAF services to discover information about which EJBROLE profiles can or cannot be authorized to. The server must be granted permission to run authorization checks against the HLQ of the EJBROLE profile name. The HLQ of the profile name is the first segment of the profile name, up to but not including the first '.'. For example:

EJBROLE profile name = "BBGZDFLT.SuperSnoop.Manager"
HLQ = "BBGZDFLT"

To grant the server permission to run authorization checks against the profile HLQ, the user associated with the server process must be granted READ access to the BBG.SECPFX.<HLQ> profile in the SERVER class:

```
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

In the example, because the SAF role mapper sets the HLQ of the mapped profile to the profilePrefix, the same profile BBG.SECPFX.BBGZDFLT governs both the APPLID authentication permissions, and the EJBROLE profile authorization permissions.



How Liberty works with RACF (cont.) WZSSAD (WLP z/OS System Security Access Domain)

- Authorizing a Subject to other SAF resources..
  - Servers can check access to other resource classes.
  - The SERVER class profile of the form: BBG.SECCLASS.classname controls this access.
  - For instance, to allow the server to check resources in the RACF FACILITY class:

```
RDEFINE SERVER BBG.SECCLASS.FACILITY UACC(NONE) OWNER(SYS1)
PERMIT BBG.SECCLASS.FACILITY CLASS(SERVER) ACCESS(READ) ID(LIBSERV)
```

The below text is reproduced from the WebSphere 8.5 InfoCenter.

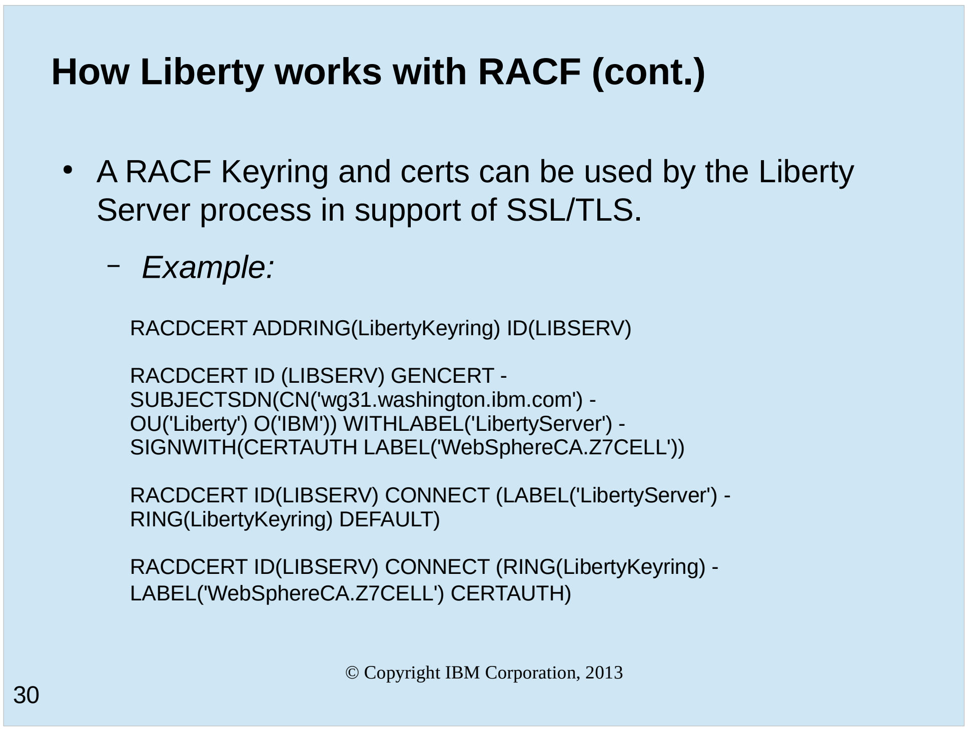
Authorizing a Subject to other SAF resources

Java EE applications can perform access control checks against SAF classes other than EJBROLE. The WZSSAD restricts which SAF classes outside of EJBROLE the server is allowed to authorize against. This prevents an unauthorized user or application from leveraging the server's use of authorized SAF services to discover information about which resource profiles in non-EJBROLE SAF classes the user or application is or is not authorized to.

To grant the server permission to perform authorization checks against non-EJBROLE SAF classes, the user associated with the server process must be granted READ access to the BBG.SECCLASS.<SAF-CLASS> profile in the SERVER class. For example, to authorize against a profile in the FACILITY class:

```
RDEFINE SERVER BBG.SECCLASS.FACILITY UACC(NONE)
PERMIT BBG.SECCLASS.FACILITY CLASS(SERVER) ACCESS(READ) ID(serverUserId)
```

Note: There are no restrictions on which resource profiles within the non-EJBROLE class the server is allowed to authorize against. If the server has been granted permission to perform authorization checks against a non-EJBROLE class, then it can authorize against any profile in that class.



How Liberty works with RACF (cont.)

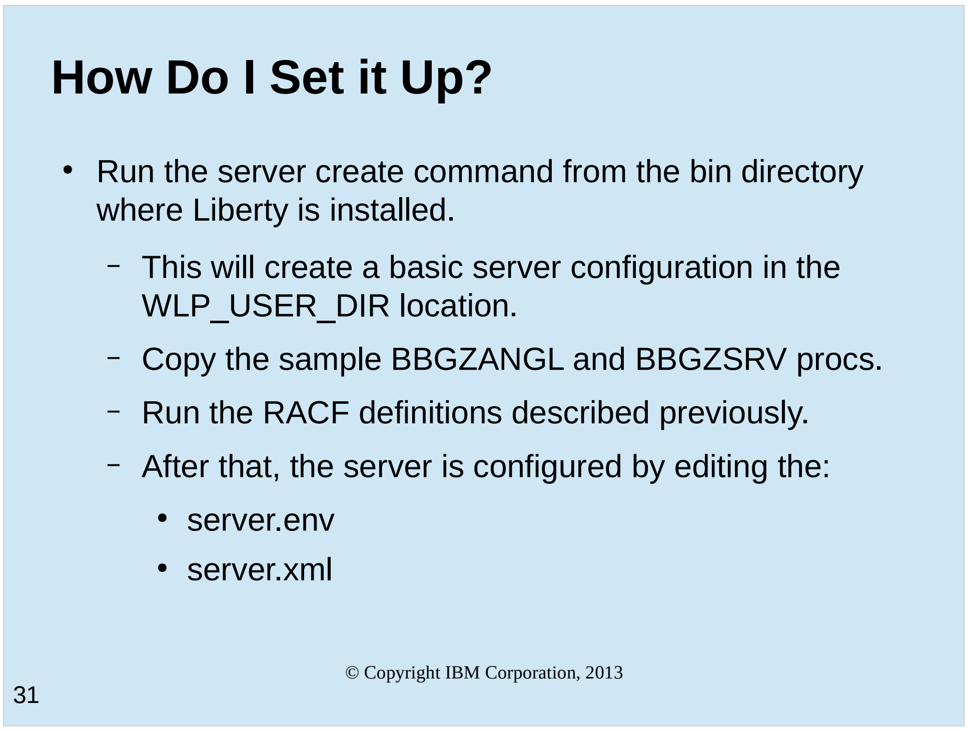
- A RACF Keyring and certs can be used by the Liberty Server process in support of SSL/TLS.
  - *Example:*

```
RACDCERT ADDRING(LibertyKeyring) ID(LIBSERV)

RACDCERT ID (LIBSERV) GENCERT -
SUBJECTSDN(CN('wg31.washington.ibm.com') -
OU('Liberty') O('IBM')) WITHLABEL('LibertyServer') -
SIGNWITH(CERTAUTH LABEL('WebSphereCA.Z7CELL'))

RACDCERT ID(LIBSERV) CONNECT (LABEL('LibertyServer') -
RING(LibertyKeyring) DEFAULT)

RACDCERT ID(LIBSERV) CONNECT (RING(LibertyKeyring) -
LABEL('WebSphereCA.Z7CELL') CERTAUTH)
```

The first command creates a keyring for the server userid.

The next command creates a personal certificate for the server userid, signed with a self-signed CERTAUTH certificate that had been previously created.

The next two commands connect the personal certificate and the CERTAUTH certificate that signed the personal certificate to the server userid's keyring.


How Do I Set it Up?

- Run the server create command from the bin directory where Liberty is installed.
  - This will create a basic server configuration in the WLP_USER_DIR location.
  - Copy the sample BBGZANGL and BBGZSRV procs.
  - Run the RACF definitions described previously.
  - After that, the server is configured by editing the:
    - server.env
    - server.xml

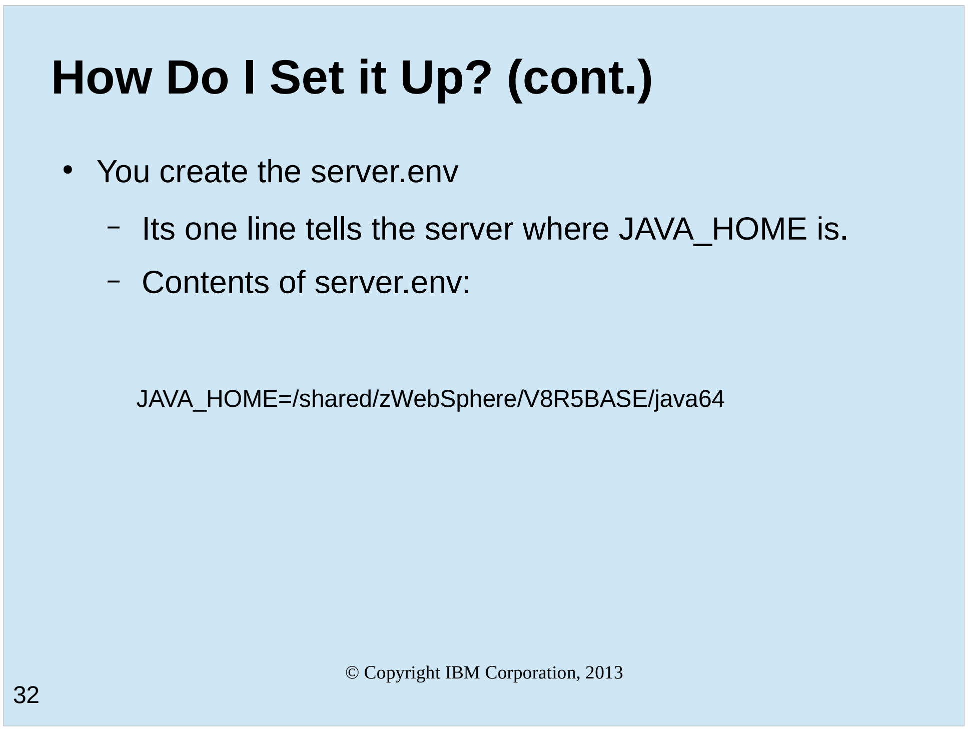
Now we are ready to set up a Liberty profile server.

Start by setting the environment variable WLP_USER_DIR to the location where you wish the server to be created.

Then copy and customize the sample PROCs for the angel and a server,, BBGZANGL and BBGZSRV.

Run the RACF definitions we have previously described.

Then you may configure the server by editing the server.env and server.xml files.



How Do I Set it Up? (cont.)

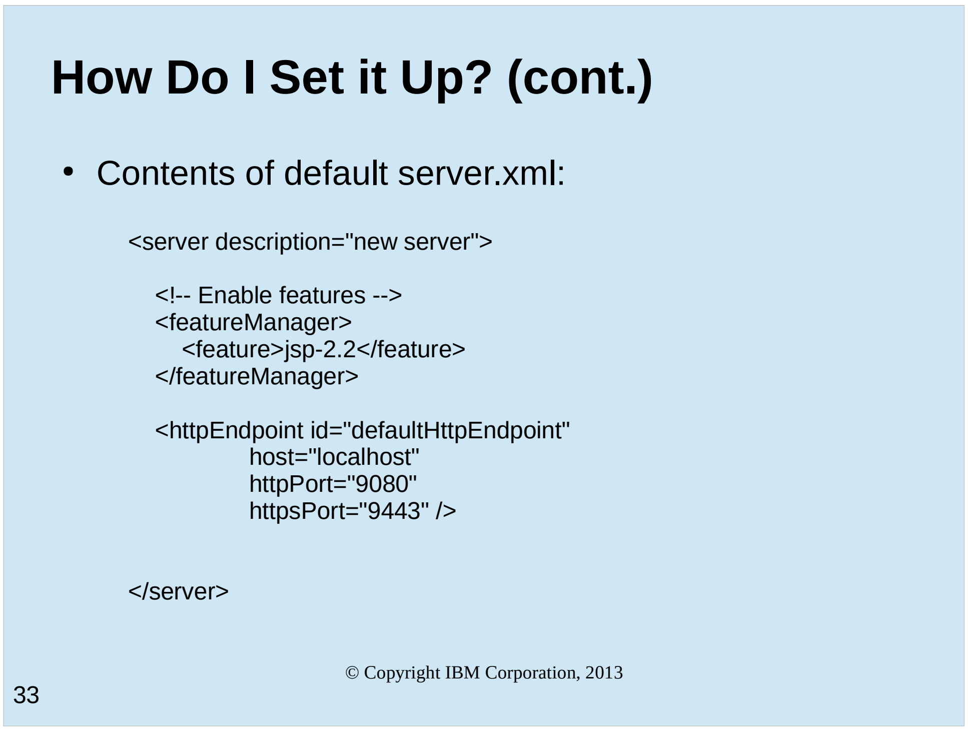
- You create the server.env
  - Its one line tells the server where JAVA_HOME is.
  - Contents of server.env:

```
JAVA_HOME=/shared/zWebSphere/V8R5BASE/java64
```

32

You'll need to create and edit the server.env file in the same directory where you find the server.xml file.

Add the JAVA_HOME environment variable pointing to a valid location for a 64bit java 6.01 or 7 SDK.

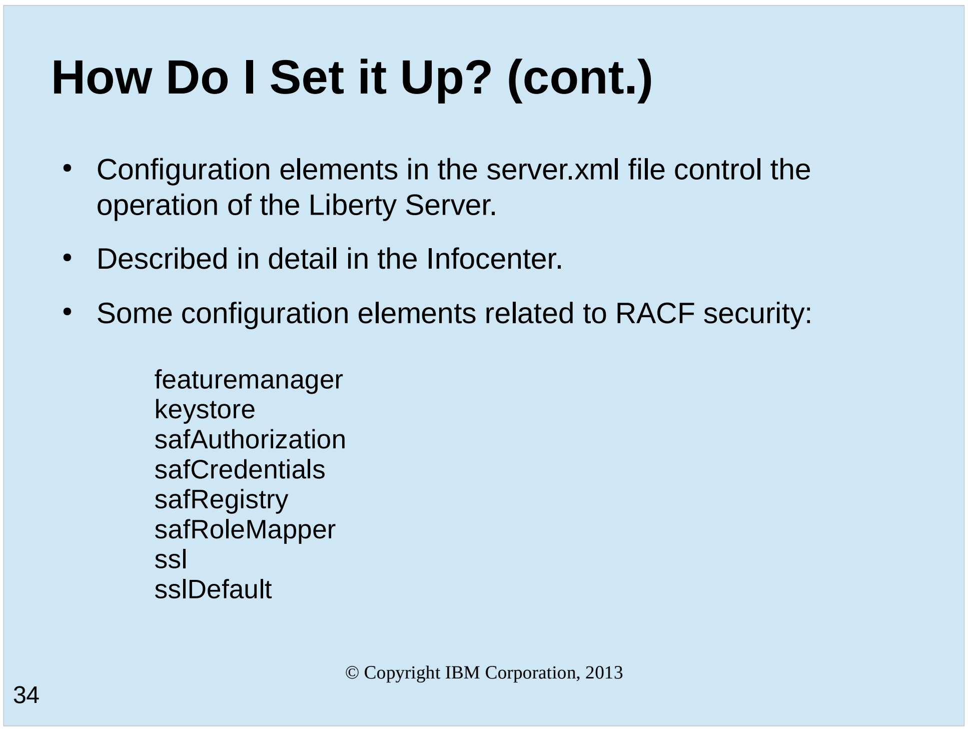


How Do I Set it Up? (cont.)

- Contents of default server.xml:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="localhost"
                  httpPort="9080"
                  httpsPort="9443" />


</server>
```

The default server.xml file that was created by the server create command is what is shown on the foil. It will allow the server to come up and stay up, but it won't do much of anything. Fast, but useless.

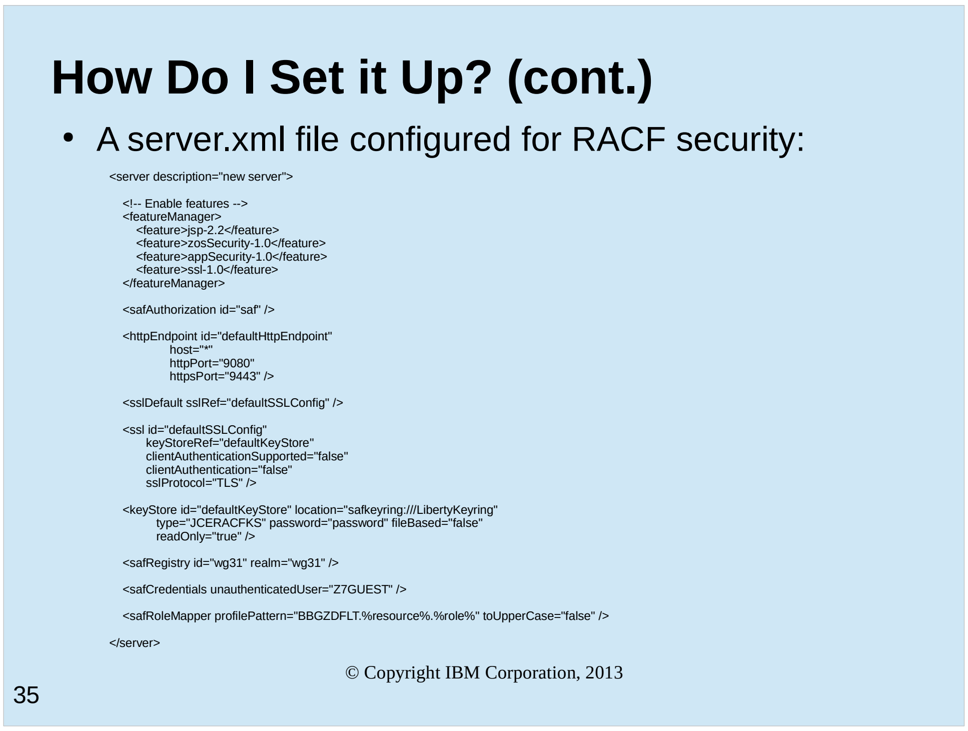


How Do I Set it Up? (cont.)

- Configuration elements in the server.xml file control the operation of the Liberty Server.
- Described in detail in the Infocenter.
- Some configuration elements related to RACF security:

    featuremanager
    keystore
    safAuthorization
    safCredentials
    safRegistry
    safRoleMapper
    ssl
    sslDefault

We'll need to edit the server.xml file to add all of the features and options that we want to use. The InfoCenter is a good reference for all of this information.

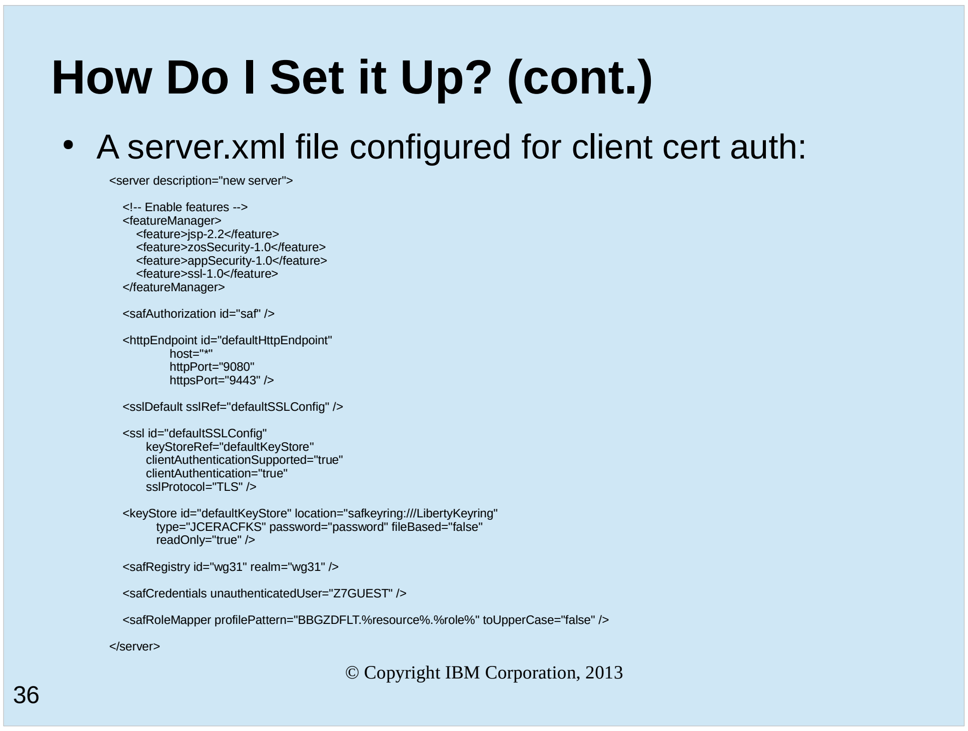


How Do I Set it Up? (cont.)

- A server.xml file configured for RACF security:

```
<server description="new server">

   <!-- Enable features -->
   <featureManager>
      <feature>jsp-2.2</feature>
      <feature>zosSecurity-1.0</feature>
      <feature>appSecurity-1.0</feature>
      <feature>ssl-1.0</feature>
   </featureManager>

   <safAuthorization id="saf" />

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

   <sslDefault sslRef="defaultSSLConfig" />

   <ssl id="defaultSSLConfig"
        keyStoreRef="defaultKeyStore"
        clientAuthenticationSupported="false"
        clientAuthentication="false"
        sslProtocol="TLS" />

   <keyStore id="defaultKeyStore" location="safkeyring:///LibertyKeyring"
          type="JCERACFKS" password="password" fileBased="false"
          readOnly="true" />

   <safRegistry id="wg31" realm="wg31" />

   <safCredentials unauthenticatedUser="Z7GUEST" />

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

</server>
```

Here are some of the things you may want to add to the server.xml file.



How Do I Set it Up? (cont.)

- A server.xml file configured for client cert auth:

```
<server description="new server">

   <!-- Enable features -->
   <featureManager>
      <feature>jsp-2.2</feature>
      <feature>zosSecurity-1.0</feature>
      <feature>appSecurity-1.0</feature>
      <feature>ssl-1.0</feature>
   </featureManager>

   <safAuthorization id="saf" />

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

   <sslDefault sslRef="defaultSSLConfig" />

   <ssl id="defaultSSLConfig"
        keyStoreRef="defaultKeyStore"
        clientAuthenticationSupported="true"
        clientAuthentication="true"
        sslProtocol="TLS" />

   <keyStore id="defaultKeyStore" location="safkeyring:///LibertyKeyring"
          type="JCERACFKS" password="password" fileBased="false"
          readOnly="true" />

   <safRegistry id="wg31" realm="wg31" />

   <safCredentials unauthenticatedUser="Z7GUEST" />

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

</server>
```

36

We are adding the zosSecurity, appSecurity, and ssl features to the featureManager component.

We are also identifying that we will be using saf for authorization.

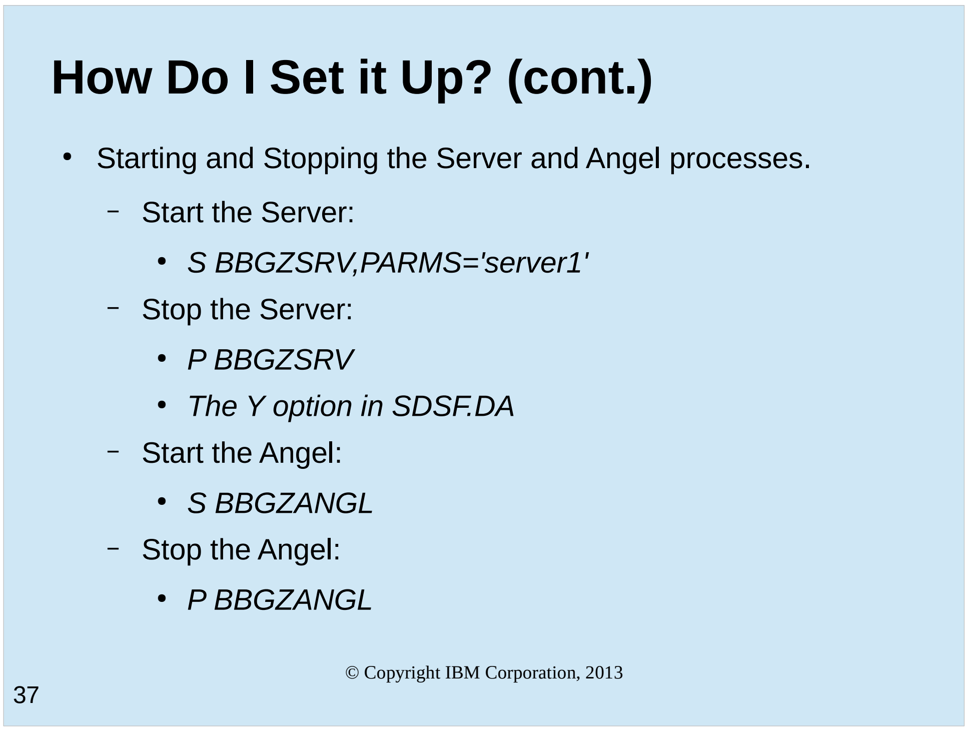
We have changed the hosts paramenter to an * (any host) from localhost (which won't work well on a z/OS system). You may need to specify an actual host name for this parameter.

We have change the ssl config so that clientAuthenticationSupported and clientAuthentication are both “true”.

We have added a keyStore that identifies the keyring we set up using the RACDCERT command.

We are specifying the safRegistry id and realm name. We are setting the unauthenticated user id to a previously set up id, (Z7GUES). This id should have very little if any authority.

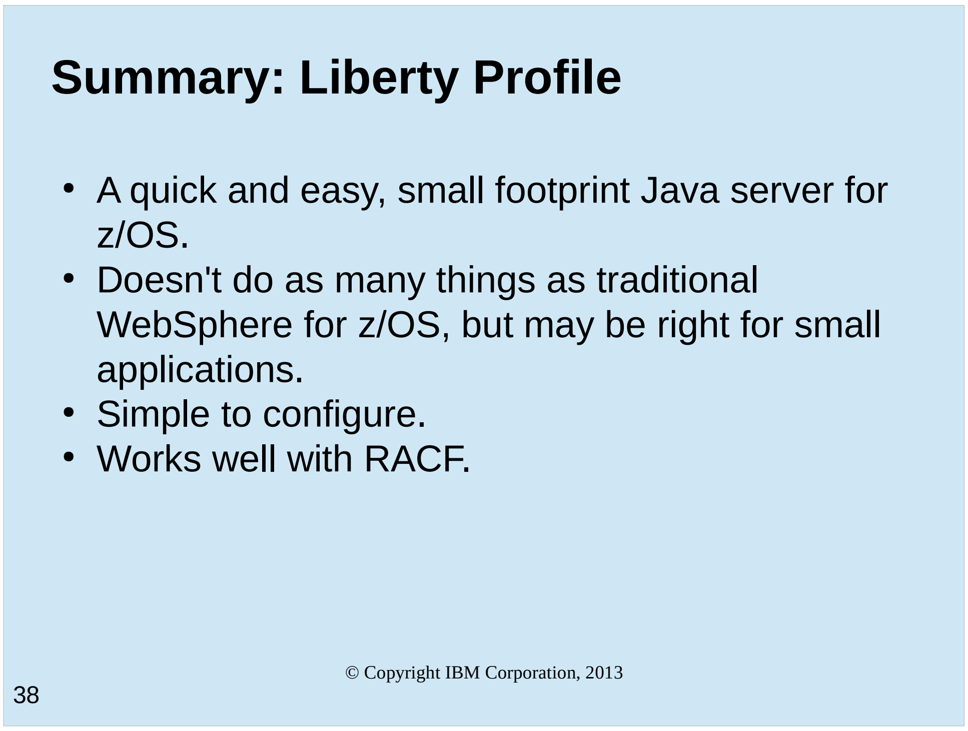
We are setting the safRoleMapper pattern to be the prefix for the hlq, the application resource name for the middle qualifier and the role name for the last qualifier, and specifying that it should not be folded to uppercase (mixed case is allowed).



How Do I Set it Up? (cont.)

- Starting and Stopping the Server and Angel processes.
  - Start the Server:
    - *S BBGZSRV,PARMS='server1'*
  - Stop the Server:
    - *P BBGZSRV*
    - *The Y option in SDSF.DA*
  - Start the Angel:
    - *S BBGZANGL*
  - Stop the Angel:
    - *P BBGZANGL*

At this point we are ready to start the server. The start command is to start the PROC with the servername as a parameter. To stop the server you just issue a stop command against the PROC name. Same for the angel process.



Summary: Liberty Profile

- A quick and easy, small footprint Java server for z/OS.
- Doesn't do as many things as traditional WebSphere for z/OS, but may be right for small applications.
- Simple to configure.
- Works well with RACF.