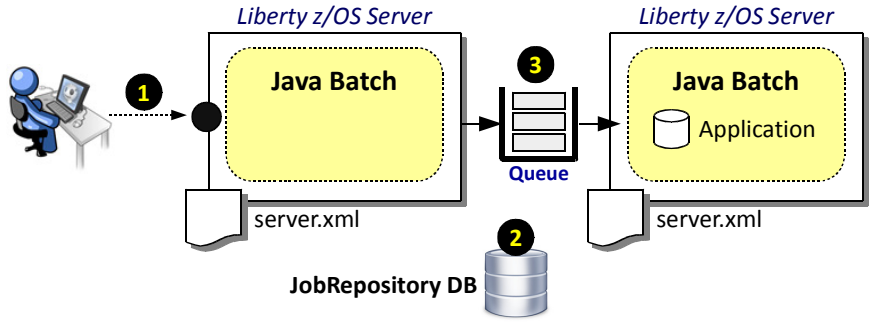
WebSphere Application Server

Unit 6

Java Batch Security

IBM WebSphere Liberty Batch z/OS

Up to This Point We've Kept Security as Simple as Possible ...



1. **Client Access**
   We simplified this by using "basic" security in server.xml for SSL, user registry, and authorization roles.
2. **DB2 Access**
   We simplified this by having the "authentication alias" on the JDBC definition be an ID that had access to the created batch tables.
3. **MQ Access**
   We simplified this by having an MQ system that does not have security enabled.

We did this to keep initial focus on the other things. But the time has come to turn our attention to the "real world" implications of security with Java Batch.

Disclaimer ...

We've reached the point in the workshop where we can't ignore security any longer. ☺

Up to this point we've kept security as simple as possible by doing a few key things:

1. The client access security, particularly network-based client access with the batchManager client, has been simplified by using the "basic security" setup in the Liberty server.xml file. This satisfied the requirements for encryption (SSL), authentication (user registry), and authorization (application roles).
2. Access to DB2 was simplified by using JDBC Type 4 (network) connection and an authentication alias ID that had fairly powerful access rights.
3. Our MQ system does not have security enabled, so we did not have to worry about granting access rights to the QMGR connections, the queues, or the topic publication or subscriptions.

All that was done to keep the workshop focused on the key topics as we worked our way through the workshop. It would have been very distracting to start out and immediately dive into security details, rather than focus on understanding the essentials of the Java Batch function, the command line interface utilities, and the multi-JVM design.

But security is not something that can be overlooked in the "real world." You have to address it eventually. For us that time has come.

Important Disclaimer

An army of lawyers ☺

What we're providing in this unit are *examples*

Always review security examples with your security administrator

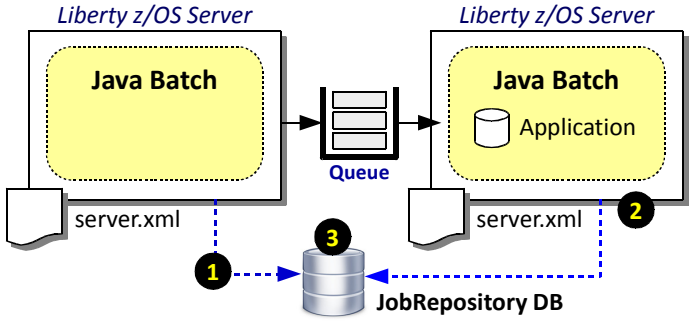
Your security policies take precedence over any examples shown here

This is a fairly important disclaimer, so even though we're using a funny picture on the chart, the message is quite serious.

Throughout this unit we are going to show examples of security commands to put in place various SAF profiles. They are only examples, and you should review any and all such security examples with your security administrator to make sure what is put in place aligns with your security policies. *Your security policies take precedence over anything shown here.*

Infrastructure

JobRepository (DB2) Access



Java Batch Tables

```
JOBINSTANCE
STEPTHREADINSTANCE
STEPTHREADEXECUTION
JOBEXECUTION
JOBPARAMETER
```

As of 16.0.0.4. This may change if new features are added that require additional tables.

1. Type of Access

If JDBC Type 2, then the ID that asserted into DB2 will the STC ID, unless you have an auth alias on the JDBC datasource.

If JDBC Type 4, then the ID that is asserted into DB2 will be the authentication alias, or a client certificate ID.

2. Who is accessing

This is related to the type of access, but it's also related to how many servers you have accessing and whether they are the same ID or different IDs.

3. GRANT on what resource?

The GRANT can be against the tables, table spaces, database, STOGROUP, buffer pools. This becomes a DB Admin question how this is arranged for your environment.

Key point: this is standard DB Admin security practices. Work with your DB Admin to set up and secure the JobRepository

Multi-JVM and queue ...

Let's first focus on the JobRepository implementation using DB2 z/OS. As mentioned, we used JDBC Type 4 with an authentication alias that had enough power to overcome any security issues. Let's step back and look at what things you and your DB2 administrator should take into account when constructing a DB2 z/OS JobRepository implementation:

1. *Type of access* -- the ID that presents itself to DB2 is going to be a function of the type of connection defined in the Liberty z/OS server. JDBC Type 2 typically means the started task ID will be asserted across the Type 2 cross-memory connection into DB2, so it is that ID that would need access to the JobRepository resources. (It's possible to code an authentication alias on a Type 2 datasource and have that alias ID asserted across, but that is not a common practice.)

   JDBC Type 4 is a network-based protocol, and that uses an authentication alias coded on the datasource element as the ID that asserted into DB2. If that's the case, then that's the ID being asserted. DB2 z/OS has recently added the ability to use a client certificate to authenticate from client to DB2 over Type 4, so if that's the mechanism you use, then the ID associated with the client certificate would need to be granted access.

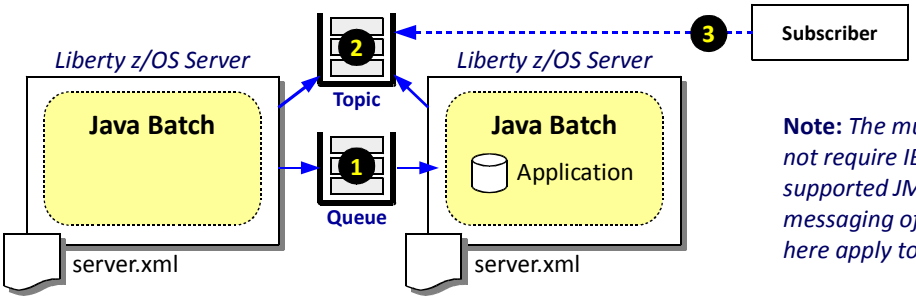
2. *Who is accessing* -- as the chart says, this is a function of the type of JDBC access, but it is also a function of how many different servers you have accessing. In a multi-JVM configuration you would have at least two servers, and perhaps more. If all your servers are accessing using the same kind of JDBC connector and with the same ID, then this is simplifed -- grant that ID the necessary access. But if the servers are asserting different IDs, then you'll have to take into account all those IDs when performing the GRANT operations on the table resources.

3. *GRANT processing* -- as of 16.0.0.4 there are five tables that comprise the JobRepository implementation. (That may change later, so always check the DDL created by the ddlGen utility to make certain.) Any ID accessing the JobRepository tables is going to need SELECT, UPDATE, INSERT, and DELETE authority. But your DB2 administrator may wish to protect access to the table spaces, storage groups and buffer pools as well. So review this with your DB2 adminsitrator to make sure the JobRepository is implemented properly and protected according to your policies.

Key Point: work with your DB2 adminsitrator to review, implement, and secure the JobRepository tables.

IBM WebSphere Liberty Batch z/OS

Multi-JVM Queueing (MQ) Access; Batch Events Topic (MQ) Access



Note: *The multi-JVM model and batch events to not require IBM MQ. They can be used with any supported JMS provider, including the default messaging of IBM Liberty. The principles outlined here apply to all. The specifics may differ.*

1. Job Submission Queue

Only in the picture if multi-JVM. This queue must allow PUT from the dispatcher ID(s), and GET from the executor ID(s). The ID is a function of the type of access: BINDINGS (cross-memory) or CLIENT (network).

2. Topic Publishers

This may be used with either a single server model or a multi-JVM model. The servers that are configured to publish events must have the authority to do so against the target QMGR.

3. Topic Subsribers

The ID of any subscriber to the topic will need to be authorized to subscribe.

batchManagerZos uses BINDINGS to access the local QMGR, so that ID is the ID that invokes the batchManagerZos client.

Key point: this is standard MQ Admin security practices. Work with your MQ Admin to set up and secure the queue and topic.

Note: we're mixing "multi-JVM" and "batch events" into the same chart even though they are not really the same thing. But they both use queueing, so that's why we're bringing them together onto the same overview chart to discuss security matters. Please understand that you can use batch events and pub/sub with a single-server configuration.

When using the multi-JVM support or the batch events support, we come up against message queuing and the security considerations involved with that. The picture is making reference to IBM MQ as the queueing mechanism because MQ is so prevalent in the market; but you could use the default messaging function of Liberty for this queuing as well. The security *concepts* for that are the same, though the specifics are different. Let's look at the considerations:

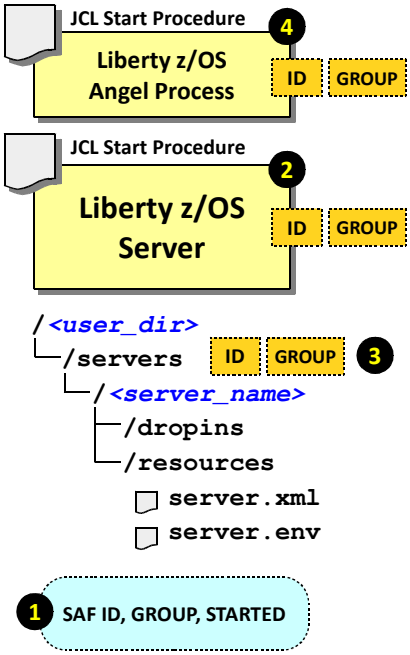
1. The job submission queue is used with the multi-JVM support. The Dispatcher server is going to access and perform PUT operations against this queue; the Executor servers are going to access and perform GET actions via their activation specifications. Like JDBC, the ID that attempts access is a function of the type of connector: MQ BINDINGS mode using a cross-memory connector and the server started task ID would be the ID that presents itself to MQ. MQ CLIENT mode is a network-based connector and that would use an authentication alias. The key here is to determine what ID is in effect and make sure that ID has the authority to access the queue manager and the job submission queue.
2. If we look at the batch events function, this involves both publishing to the topic, and subscribing to the topic. Any WebSphere Liberty Java Batch server can be made to be a publisher of events. That includes dispatcher servers, executor servers, or servers that are not part of a multi-JVM design. (For that matter, it's possible that Liberty servers on other platforms could be publishers to the MQ topic.) For z/OS servers and z/OS MQ, the access could be either BINDINGS or CLIENT, and the ID that presents itself is a function of that. (Servers on other platforms will come over CLIENT mode access.) The key here is to determine what ID is presenting itself and to make sure it has the authority to access and publish events to the topic.
3. Batch events subscribers could be any function on any platform capable of accessing and subscribing to the topic. So the set of potential IDs needing security access is larger. The most likely case is these subscribers would come into MQ using CLIENT mode, and the ID they would present would be based on the alias they have defined.

   One subscriber of note is the batchManagerZos command line client when it uses the --queueManagerName and --wait options on the submit action. That tells batchManagerZos to subscribe to the topic and watch for one of three events to determine the job status (see Unit 4 for more on this). That function uses BINDINGS mode, and thus it imposes a requirement that the QMGR be on the same LPAR as the batchManagerZos client. The ID that will present to MQ will be the ID under which batchManagerZos is invoked. That is either the ID of the authenticated user of the shell (Telnet, SSH, or OMVS), or the effective ID of the JCL job that uses BPXBATCH to invoke batchManagerZos.

Key point: work with your MQ administrator to make sure the security access policies are in place to allow either multi-JVM job submission or MDB message selection, or batch events pub/sub, to the MQ environment.

IBM WebSphere Liberty Batch z/OS

Liberty z/OS Server Infrastructure Security



1. Essential SAF Profiles

These include the ID definitions, group definitions, and the STARTED profiles to assign the IDs to the groups for started tasks.

2. Server STC ID and GROUP

The server is going to be assigned an ID, and it will have a group. There's a relationship between this ID/group and the configuration file ID/group because the server ID will need to **read** configuration files, and **write** output files.

3. Configuration file owner and group

By default this will be set to the ID/group of the userid that creates the server. You may have that ID be the same as the server STC ID/group. That is the easiest approach. But it may not be the "best," depending on your security policies. So you may require the file ownership to be different from the STC ID. Another consideration is how others will manage (delete, trim, update) the configuration files. Do you grant 'write' to the 'other' permission bits?

It's a big topic that's covered more thoroughly under the WP102687 Techdoc "Security" section. For this workshop we went with the easy approach: STC ID = file system owner ID, and your TSO ID has sufficient authority to read/write the files.

4. Angel Process

The Angel is needed when z/OS authorized services are employed. For Java Batch, that implies the batchManagerZos client. We'll defer this to that section of the unit.

batchManagerZos ...

Then there's the question of the Liberty z/OS infrastructure security, which we'll focus on here.

Note: we're differentiating "infrastructure" from "application" security. Application security would be that required to allow batchManager and REST clients access, as well as AdminCenter access. The line between "infrastructure" and "application" security is not a clear, bright line however. But for now we'll draw a line and talk about "infrastructure" as what's shown on this chart.

There are four areas to consider:

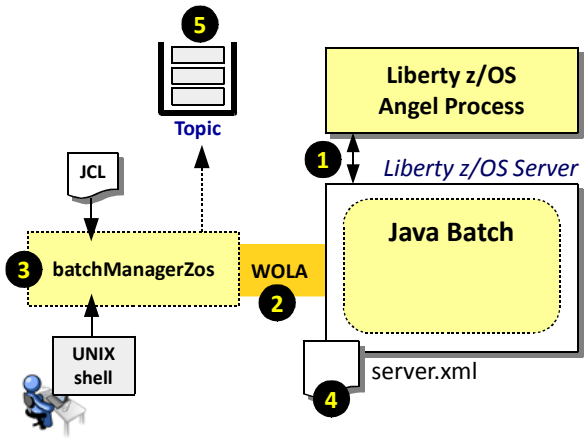
1. *Essential SAF profiles* -- before you can start a server and assign an ID to it, that ID must be defined. That ID will be connected to a group. The STARTED profiles need to be in place to make the assignment based on the start procedure and perhaps JOBNAME. When we refer to "essential SAF profiles," this what we're referring to.
2. *Server STC ID and GROUP* -- the Liberty z/OS server will be assigned an ID based on the STARTED profile that is matched with the START command. The ID that's assigned to the server has a relationship to other things. We noted for JDBC T2 and MQ BINDINGS mode, that will be the ID that's asserted. But it's also the ID that will seek access to the configuration file system for both READ and WRITE operations.
3. *Configuration file owner and group* -- a consideration is whether the STC ID is the same as the file owning ID. They *can* be -- that's the simplest approach -- but they don't *have* to be. If they're the same, then the WRITE operations to the configuration file system is easily accomplished, but if they're different then you'll have to use the WLP_OUTPUT_DIR variable to direct server output to a different location where the STC ID does have write permissions (or grant the 'group' or 'other' write permissions, but that's not a good practice). This opens up a broader topic which we'll defer from this workshop. We'll point you at the WP102687 Techdoc, which has a unit on security where this issue of STC ID and file owning ID is covered in greater detail.
4. *Angel Process* -- as we've seen, this comes into play when any Liberty z/OS server seeks to use z/OS authorized services. For Java Batch, the Angel is needed to use the batchManagerZos command line client (because that uses WOLA), and if SAF is used as the user registry and for authorization role enforcement. The Angel Process is a started task, and as such has its own ID and group and STARTED profiles to assign the ID to the started task. It also brings SERVER profiles to the table which allow the Liberty z/OS servers access to the various authorized services. We see that later in this unit.

Client Access

batchManagerZos

IBM WebSphere Liberty Batch z/OS

batchManagerZos, WOLA, the Angel Process, and Topic Subscription



1. **Server access to Angel**
   This is controlled by SAF SERVER profiles. Details on that coming up.
2. **WOLA registration into server**
   This is controlled by a SAF CBIND profile, which is based on the WOLA three part name configured in the server.xml. The ID under which batchManagerZos runs needs READ to that CBIND.
3. **The ID under which batchManagerZos runs**
   This is determined by *how* the utility is invoked. If from a UNIX shell environment, then the ID in the shell environment is used. If by submitted JCL (using BPXBATCH), then it's the effective ID of the job: either the submitter's ID, or USER= on the JOB card.
4. **Basic vs. SAF authentication**
   If you have "basic" in effect, then the batchManagerZos ID can't be determined, which is why we used SPECIAL-SUBJECT=EVERYONE earlier.
   If you have SAF authentication, then it can validate the ID and use that for further role checking.
5. **batchManagerZos --queueManagerName**
   This is optional, but if used then ID under which batchManagerZos runs will be asserted over BINDINGS mode to the named QMGR. That ID must be given authority to connect *and* to subscribe to the topic.

Detour: named angels ...

We're going to start with batchManagerZos because in some ways that's simpler than batchManager (though in other ways it has its own security challenges). We'll look at five areas of security for a batchManagerZos job submission:

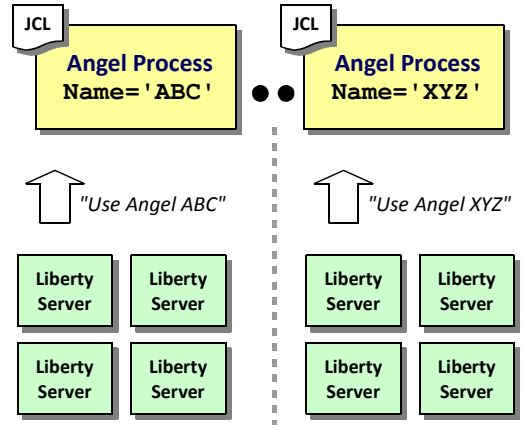
1. Before the Liberty z/OS server can have access to WOLA, which is a z/OS authorized service, it must have access to the SERVER profiles that protect that access. We'll provide details of those SERVER profiles.
2. The batchManagerZos client will create what's called a "WOLA registration" into the Liberty z/OS server. That's the cross-memory connection WOLA uses. A SAF CBIND profile protects what client IDs are allowed to do this, and that CBIND is based on the WOLA three-part name that's defined in the server.xml file. The ID of the client attempting to create the WOLA registration into the server must have READ to the CBIND based on that three-part name.
3. batchManagerZos is a native z/OS client program, and the ID it will run under is a function of how it is invoked. If the client is invoked from a logged-in shell environment (such as a Telnet session), then batchManagerZos runs under that ID. But if you are invoking batchManagerZos using BPXBATCH inside JCL, then the ID will be whatever ID is in effect for that submitted job. That could be controlled with USER= on the JOB card, or there are other z/OS mechanisms to control the effective ID for a job.

   This is important to understand for two reasons: (1) the ability to create the WOLA registration into the server is based on that ID's access to the CBIND profile, and (2) that ID is going to need access to the application role that protects the batch function.
4. If you're using the "basic" security setup in server.xml then we need to add something to the application role definition there to let the batchManagerZos ID have access. That's because the *basic* security can't determine the ID coming over WOLA, so the ID is seen as "null." If you're using SAF-based authentication then it *can* determine the ID, and then the batchManagerZos ID will need to be on the application role to permit access to the batch function.
5. Finally, if you're using --queueManagerName on the batchManagerZos command line, that implies a BINDINGS mode connection to the local QMGR to subscribe to the batch topic. That ID must be granted access to the QMGR and the topic, as we discussed on the earlier chart regarding MQ.

We'll get into the details of some fo this in a few charts, but first let's take a detour and look at something called "named Angels," which is support that came in with the 16.0.0.4 release.

IBM WebSphere Liberty Batch z/OS

Slight Detour: "Named Angels" introduced in 16.0.0.4



bootstrap.properties

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

Before: one Angel per LPAR

Liberty z/OS prior to 16.0.0.4 was limited to a single Angel per LPAR. If that Angel needed to be stopped, all servers on the LPAR using that Angel were affected. The ability to isolate was needed.

Now: ability to start multiple "named" Angels

With 16.0.0.4 the Angel code was updated to take a name=' ' parameter on the PROC statement. The Angel starts with a name. Servers can be told to use a specific "named" Angel.

bootstrap.properties file

We've not mentioned this file before now. This file names properties set when the Liberty server starts initially. Two new properties can be set: use named Angel, and the name of the Angel to use. If these properties are absent, the server will seek the one "unnamed" Angel.

Why we mention this

Because the SAF SERVER profiles you create and grant the server ID READ to is affected by this. On the next chart we'll show you those SERVER profiles, and you'll see the "named Angel" effect.

The 16.0.0.4 release of Liberty z/OS brought something called "named Angels" into the picture. To explain what this does it's important to provide just a little history so you understand *why* it was created.

Prior to the named Angel support there was only one Angel per LPAR, and all Liberty z/OS servers on the LPAR made use of that one Angel. That works as long as (a) the Angel stays up all the time, and (b) the level of code the Angel uses doesn't require update too frequently. But what we saw was neither "a" nor "b" was the case, and a requirement surfaced to allow groups to have their own Angel process they could control. One group did not want to be affected by another group dropping their Angel.

So "named Angels" came into the picture. This allows the existence of multiple Angels on a given LPAR. To keep one Angel separate from other Angels they now carry a name. That name is provided on the Angel JCL start procedure PROC statement.

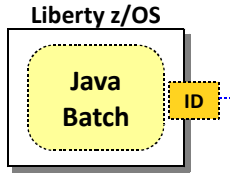
Note: it's possible to have a mixture of one "unnamed Angel" and some number of "named Angels" on an LPAR. The unnamed would be considered the "default" Angel while the others would be used by specific groups that pointed their servers at their named Angel.

To have a Liberty z/OS server use a specific, named Angel you provide two lines in the bootstrap.properties file that's in effect for the server. Those properties are shown on the chart. The first property turns on the use of named Angels for the server, the second property names the Angel to use. If the Angel that is named is not present, then the Liberty z/OS server will not get access to authorized services.

The reason we bring this to your attention is because we're about to show the SAF SERVER profiles that must be in effect for a server to get access to z/OS authorized services. How you structure one of the SERVER profiles is dependent on whether you are using named Angels.

SAF SERVER Profiles to Allow the Liberty z/OS Server to Use WOLA

If you're using a named Angel, then make sure your server ID has read to the BBG.ANGEL.*<angel_name>* profile. Otherwise, READ to the BBG.ANGEL profile.



*The batchManagerZos ID does **not** need READ to these profiles. It does, however, need READ to the CBIND to allow it to "register" into the server.*

`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*

To make things easy you could create a GROUP that has READ to these profiles, then simply connect server IDs to the group.

SSL and authentication? ...

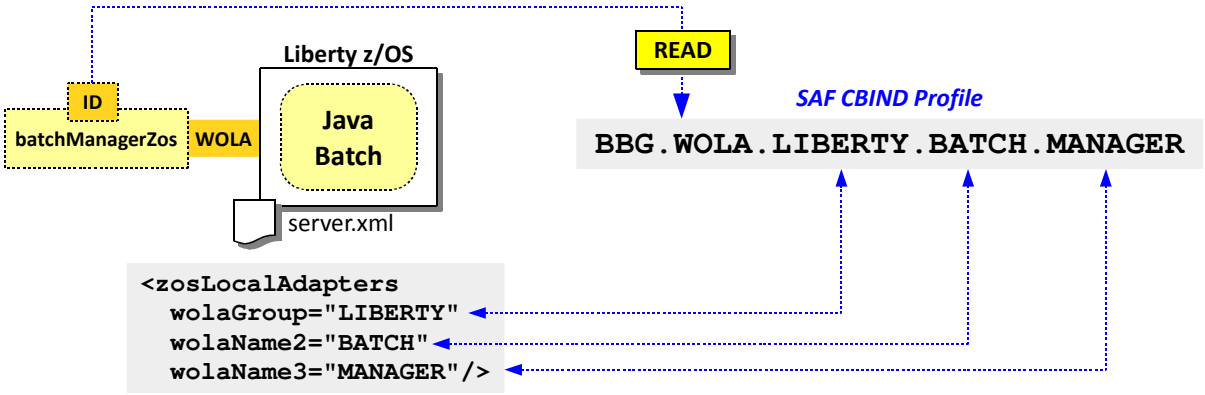
Here are the SAF SERVER profiles needed so a Liberty z/OS server can get access to WOLA authorized services. The ones highlighted in yellow are required for WOLA; the ones in gray are for other authorized services:

- BBG.ANGEL or BBG.ANGEL.<angel_name> -- this is why we had the previous chart: depending on whether you're using an unnamed Angel, or a named Angel, you will need the proper SERVER profile with your Liberty z/OS server STC ID granted READ to the profile.
- BBG.AUTHMOD.BBGZSAFM and BBGZSCFM -- these do *not* have <angel_name> as part of their names; your Liberty z/OS server STC ID needs READ to both of these.
- BBG.AUTHMOD.BBGZSAFM.LOCALCOM, WOLA, and BBGZSCFM.WOLA -- your server STC ID needs READ to all three of those.

The ID under which the batchManagerZos client operates does not need read to the SAF server profiles. It does require READ to the CBIND profile, however. Otherwise, the WOLA registration into the Liberty z/OS server will not succeed. That's next.

IBM WebSphere Liberty Batch z/OS

SAF CBIND Profile and the WOLA Three Part Name



If the ID does not have READ to the CBIND, you will get this error:

```
ERROR: [jnu_wolaGetConnection(_CHAR12 *, int, _CHAR12 *)] rc=12  BBOA1CNG RC:12, RSN:14,
  registrationName:13b681ab batchManager, waitTime:30, connHandle:13b681b7
ERROR: [jnu_initAndOpenWolaConn(WolaConn *, char *)] rc=12  after jnu_openConn
ERROR: [jnu_newBatchWolaClient(cJSON *)] rc=12  after jnu_initAndOpenWolaConn
ERROR: [jnu_main(int, char **)] rc=255  exit: jnu_newBatchWolaClient returned NULL
```


12

SSL and authentication? ...

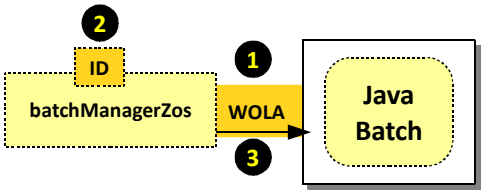
The CBIND profile is what prevents just anyone from firing up an instance of batchManagerZos and connecting into your server and trying to submit a batch job. The ID under which the batchManagerZos client is operating must have READ to the SAF CBIND profile that matches the WOLA three-part name that's defined in your server.

The picture above illustrates the relationship. The WOLA three-part name for the server is defined using the <zosLocalAdapters> element in the server.xml. The CBIND profile has the three part name as part of its construction. The batchManagerZos ID must have READ to that profile. If it does not have READ, the client will get the error shown at the bottom of the chart.

Note: it is possible to use wildcard characters for the CBIND profile. That would allow you to have a few, or even one, CBIND profile that would work with a number of Liberty z/OS servers with different three-part names. It's also possible to grant a group the READ access to the profile, then connect server IDs to that group. That would alleviate the need to go back and grant access to the CBIND every time a new server comes online.

IBM WebSphere Liberty Batch z/OS

What About WOLA Data Encryption (SSL) and User Authentication?



1. Cross-memory; no SSL needed

WOLA is a cross-memory technology; there is no network involved. The cross-memory communication between batchMangagerZos and the Liberty z/OS server can't be "sniffed." There is no need to encrypt.

2. Already authenticated to z/OS

batchManagerZos is a z/OS-native utility that can only be invoked on z/OS. To invoke it you have already either logged into OMVS, or you opened a Telnet or SSH session to the mainframe. The z/OS system knows who you are. If it is invoked through submitted JCL, then the effective ID for that job submission applies, and that ID is either good or not good.

3. WOLA connection requires READ to CBIND

The batchManagerZos ID will only be able to connect using WOLA if that ID has READ to the appropriate CBIND profile. That's further validation that the ID is good and permitted.

Authorization **to use Java Batch is another issue … we look at that next**

Let's step back and ask a few high-level questions about security and batchManagerZos:

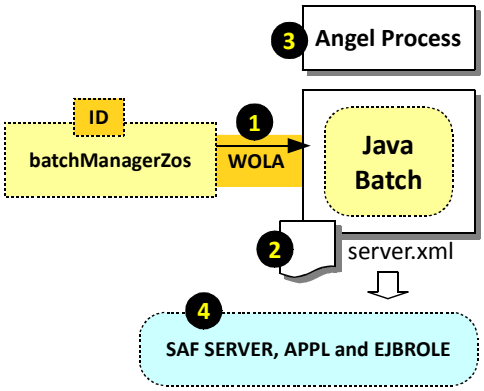
- What about SSL? Do we need to worry about that? -- answer: no. batchManagerZos uses WOLA, which is a cross-memory exchange mechanism that uses no network at all. There's no need for encryption because there's no network over which any traffic flows. The cross-memory connection can't be "sniffed."
- What about authentication? There's no --user and --password on the batchManagerZos command. -- answer: there's no authentication between batchManagerZos and the server it connects to. We know the ID that invokes batchManagerZos is a valid z/OS ID, otherwise we wouldn't be able to invoke batchManagerZos. If we invoke batchManagerZos from a command prompt, then we would have had to log into the shell environment, and that would authenticate us to z/OS. If we invoke batchManagerZos from JCL, then the ID of the submitted JCL is authenticated by JES, otherwise the JCL job would not run.

  Also, we just saw how the CBIND profile protects the server from having unauthorized IDs connect with WOLA into the server, so that's *more* validation of the ID and its authority to connect.

The batchManagerZos client is a very secure client to use because of the way WOLA works, and the way WOLA usage is protected overall.

IBM WebSphere Liberty Batch z/OS

What About WOLA Authorization?



1. **The ID of batchManagerZos will be asserted over WOLA**
   There is no ID/password specified on the batchManagerZos command line. The ID that will be asserted in will be the ID under which the batchManagerZos client is operating.
2. **If WOLA, then use SAF registry and authorization**
   Long story short: "basic" security does not have visibility to the ID that's asserted over WOLA. Since it can't know the ID, then it can't authorize the ID. That's why we used SPECIAL-SUBJECT=EVERYONE earlier.
   If we wish to get visibility to the asserted ID, we need to tell Liberty z/OS to use SAF for the registry. Then we get visibility, and from there we can check against SAF roles for authorization. Enabling SAF involves XML updates to the server.xml file.
3. **The Angel is needed for SAF use**
   The Angel is needed for WOLA, and it's also needed for SAF. The server ID needs READ to the **SAFCRED** profile as well as the WOLA profiles.
4. **SAF profiles in support of SAF registry and authorization**
   There's a handful of SAF updates needed to support role authorization.

XML updates and SAF ...

We still have to consider *authorization*, which is the checking to see if the ID that's been presented has the authority to access the WebSphere Liberty Java Batch function. That ID, as we discussed, will be the ID under which batchManagerZos is invoked, and that will be asserted over the WOLA connection into the Liberty z/OS server.

We mentioned how the "basic security" setup didn't work too well with WOLA and authorization because the ID that was asserted over WOLA was seen as "null" by the basic security setup, and therefore it could not check the basic authorization roles for access authority. That's why we had to code the SPECIAL-SUBJECT=EVERYONE, which says any ID has access. Clearly that's not a good practice. It's what we had to do to make batchManagerZos work with basic security, but you would not do that in the real world.

What you would do is move security out of "basic" and down into SAF, and once you do that the ID that comes over WOLA can be seen and properly authorized. This triggers the need for the Angel, but you already needed the Angel for WOLA, so what this really means is access to just one more SERVER profile for your server ID. That one extra SERVER profile you need access to is this:

`BBG.AUTHMOD.BBGZSAFM.SAFCRED`

That would be *in addition* to the SERVER profiles we already discussed for WOLA.

There are some updates to server.xml to move the security processing down into SAF, and we'll cover those next.

IBM WebSphere Liberty Batch z/OS

XML updates and SAF Updates for SAF Registry and SAF Authorization

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

<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials profilePrefix="BBGZDFLT" />

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

<basicRegistry id="basic1" realm="jbatch">
  <user name="Fred" password="fredpwd" />
</basicRegistry>

<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <special-subject type="EVERYONE"/>
    <user name="Fred" />
  </security-role>
</authorization-roles>
```

Add to the feature list

Add these three lines

The "profilePrefix" value may be any string you want. Showing default here. Key is be consistent with the EJBROLE prefix used (see below).

Leave the "basic" keystore (for now)

WOLA doesn't care, but others may be using it.

Remove basic registry and basic roles

Define APPL for profile prefix

Define EJBROLE (case sensitive)

RACLIST REFRESH as needed

```
RDEF APPL BBGZDFLT OWNER(SYS1) UACC(READ)

RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchAdmin OWNER(SYS1) UACC(NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchSubmitter OWNER(SYS1) UACC(NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchMonitor OWNER(SYS1) UACC(NONE)
```

Grant ID of your batchManagerZos user to the profile for access desired

batchManager ...

15

Our objective here is to get rid of the "basic security" setup and move the registry and role authorization down into SAF. This is accomplished with the following actions:

- The zosSecurity-1.0 feature needs to be added to the feature list
- You need to add the three lines shown on the chart:

  <safRegistry> tells Liberty z/OS to go to SAF when it needs to see if an ID is valid.

  <safAuthorization> tells Liberty z/OS to go to SAF to check application role access, specifically EJBROLE access. The racRouteLog="ASIS" attribute we have in there so ICH408I messages come out if for some reason the checks against EJBROLE fails. The ICH408I error messages will help you debug the problem. (Those are RACF messages, if you have some other SAF security product, then check with your security administrator for assistance in debugging issues with failures to gain application authority via EJBROLE definitions.)

  <safCredentials> defines the "security prefix," which will be found on the EJROLE profiles that are used for this server.
- Leave the "basic" <keyStore> definition in for now. WOLA does not use that, but someone else might need that.
- Remove the other "basic" security definitions.
- Add a SAF APPL profile for the security prefix value you have defined on <safCredentials>. We're showing BBGZDFLT, which is the default value. You can code any valid 8-character string you wish.
- Then create the three EJBROLE definitions -- one for batchAdmin, one for batchSubmitter, and one for batchMonitor -- making sure the prefix you use is the same as the APPL you create, and matches the profilePrefix= value you specified.
- Finally grant the batchManagerZos ID READ to whichever EJBROLE matches the authority you want that ID to have when it gets into the Java Batch function.

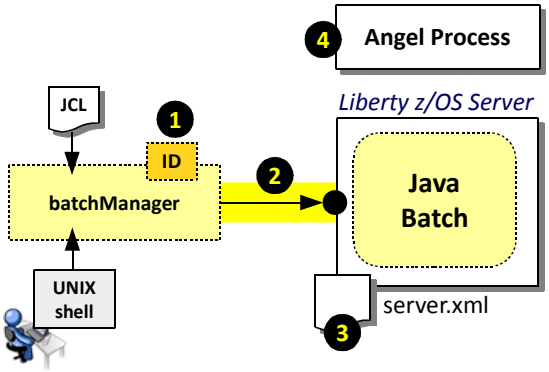
That provides the Liberty z/OS server an understanding of where it is to go for authentication and authorization, and it provides the SAF profile support for it to use SAF for those purposes.

Client Access

batchManager, including SAF SSL

IBM WebSphere Liberty Batch z/OS

batchManager and Things to Consider when Changing from "Basic" to SAF



What follows in this section is what would be required for the REST interface as well.

The batchManager client uses the REST interface of Liberty when it operates.

1. batchManager ID that flows

This is *not* based on the ID that invokes the client, it is based on either the user/password on the command line, or the client certificate that flows.

Note: if using SAF registry, the ID that flows must be defined in SAF.

2. Network connection and SSL

batchManager is a network-based client, and the connection will be redirected to SSL, which implies certificates. The "basic" security setup used a Liberty-generated certificate, but a more "real-world" scenario would use a generated server certificate signed by a well-known CA. Further, that certificate would be held in SAF keyrings.

3. Changes to server.xml

To move from "basic" security to SAF implies a few new features, the removal of the "basic" elements and the addition of elements for SAF registry, authorization, and SSL. It's similar to what we saw for batchManagerZos, but with some additional things for SSL and for the "unauthenticated userid."

4. Angel Process

The Angel is not needed for batchManager per se, or for SAF keyring SSL, but it *is* required if we use SAF for role authorization.

Detour: how SSL works ...

batchManager is different from batchManagerZos in that is uses a network connection with the server rather than WOLA. That means the security requirements are different. Let's review the considerations:

1. The ID that flows from batchManager up to the Liberty server is based on the --user value supplied on the batchManager command. The ID is *not* related to how the client is invoked; if invoked from the shell or from JCL BPXBATCH, the ID will be what's coded on the --user option value.

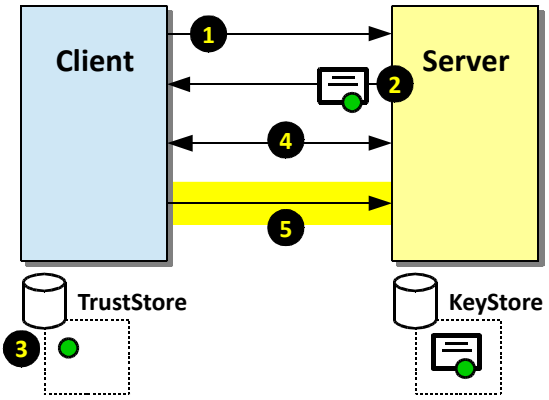
   Note: that's not entirely true ... if you're using SAF for SSL you can also use a client certificate to authenticate, which removes the need to code --user and --pasword on the command. The certificate that flows up then gets mapped to a valid SAF ID.

2. batchManager actually uses the REST interface of WebSphere Liberty Java Batch, and that interface is marked protected, which means the data link must be encrypted. It will enforce use of SSL. That means the requirements of SSL must be present, which involves digital certificates. We went the simple route with "basic" security, which produced a self-signed certificate created by Liberty, but in the real world you'd have a digital certificate signed by a well-known Certificate Authority, and you'd store that certificate in a file-based keystore on in a SAF keyring. We're going to illustrate SAF keyring.
3. In the batchManagerZos section we showed the server.xml updates to move basic security out and SAF security in. The changes for batchManager are very similar, with a few more things needed to support pushing the SSL support down into SAF keyrings. Because we're coming in on a network connection we also have to take into consideration something called the "unauthenticated user," which we'll explain a bit later.
4. The Angel Process is not needed when batchManager is used, but it *is* needed if use SAF for authentication and authorization. The updates are the same -- access to the SAFCRED SERVER profile by the server STC ID.

Before we get into the details we'll take a brief detour and look at how SSL works.

IBM WebSphere Liberty Batch z/OS

Brief Detour: How TLS (aka, "SSL") Works with Certificates and CA Signers



The Key and Trust stores may be file-based or SAF keyrings. We're getting ready to illustrate SAF keyrings.

1. **Client attempts connection with Server**
   The client (batchManager in our scenario) initiates the conversation with a connection request to the server. The Java Batch interface in Liberty is protected, so it redirects to SSL
2. **Server sends its certificate, signed by CA**
   The server retrieves its "server certificate" from its KeyStore and sends it to the client. The certificate has been signed by a "Certificate Authority" (CA). This is the process by which the client comes to "trust" the server is who the server is supposed to be.
3. **Client compares server with CAs it knows about***
   The client receives the server certificate and inspects it. It sees the certificate is signed by a CA, so it looks in its TrustStore to see if it has a matching "public key" for that CA. If it has a match, then it "trusts" the certificate and therefore the server. If no match, then it throws a security challenge. This is the browser challenge thing we've all seen.
4. **They negotiate encryption keys**
   The seek to find the strongest encryption both support.
5. **SSL connection established**
   When they agree, the SSL connection is established and the data is encrypted using the agreed-to keys.

* The `-trustSslCertificates` parameter on the batchManager command tells batchManager to trust the certificate regardless.

Transport Layer Security (TLS, but almost everyone still refers to this as SSL) is a mechanism for encrypting communications on a network between a client and a server. It uses digital certifcates as part of this process, and that's what takes us into the realm of discussions of keystores, truststores, and SAF keyrings.

1. The client initiates the conversation by sending a request up to the server. If this is to the HTTP port and SSL is required, the server will redirect the request to the HTTPS port.
2. The server will then send it's "server certificate" down to the client so the client can be certain it is talking to a server that can be trusted. This certificate is kept in the server's "keystore" (which can be a file or it can be a SAF keyring). That certificate is typically "signed" by a Certificate Authority (CA), which vouches for the authenticity of the server certificate.
3. The client receives the server certificate and then attempts to determine if the certificate is authentic. It does this by inspecting the certificate to make sure it is signed by a CA, and then if the CA that signed it is one the client knows about.

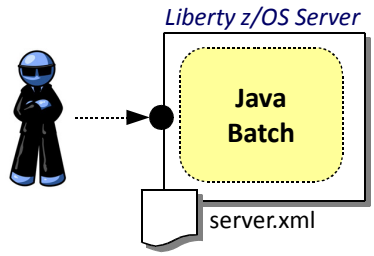
   Note: this was the "problem" with the Liberty-generated certificate we created with "basic" security -- it was a self-signed certificate, which means there was no CA that signed it. We had to get around that with the --trustSslCertificates option on the batchManager command so it would ignore this "problem."

   Even if it's signed by a CA, the client has to verify the validity of the CA's signature. It does this with a copy of the CA's public key, which is needs to have in its trust store. If the CA public key is there, the server certificate is validated and the SSL session can be established. If the CA public key is not there, then a challenge is issued (you've likely seen this in your browser when some website has a certificate the browser -- which comes loaded with a bunch of well-known CA public keys -- doesn't recognize).

   Note: the --trustSslCertificates option would get around an unknown CA issue as well.
4. Once the server's certificate is accepted the two sides negotiate with one another to find the highest level of encryption both sides support.
5. When they settle on the agreed-to level of encryption, the SSL session is built and all traffic then flows over that encrypted link.

We're going to take one more detour ... this time into the world of the "unauthenticated user."

Another Brief Detour: The Unauthenticated User



When coming in on a network connection using SAF registry, there's a *brief* period of time between the user's entry and that user being properly authenticated.

During that brief period they are *unauthenticated*. They're on a thread in the server … and we don't yet know who they are.

We need to make sure they are *powerless* during that brief time.

```
ADDGROUP WSGUESTG OMVS(AUTOGID) OWNER(SYS1)
ADDUSER WSGUEST RESTRICTED DFLTGRP(WSGUESTG) OMVS(AUTOUID –
HOME(/u/wsguest) PROGRAM(/bin/sh)) –
NAME('UNAUTHENTICATED USER') NOPASSWORD NOOIDCARD

PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(WSGUEST)
RALT APPL BBGZDFLT UACC(READ)
```

Separate group for the unauthenticated user

RESTRICTED means the user ID cannot be used to access protected resources they are not specifically authorized to access.

NOPASSWORD, NOOIDCARD means the ID can't be used to log onto the system.

Unauthenticated must have READ to the APPL that defines the profile prefix.

```
<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials unauthenticatedUser="WSGUEST"
  profilePrefix="BBGZDFLT" />
```

This is the server.xml we saw for batchManagerZos, but with the unauthenticated user specified. The default is "WSGUEST."

SAF work ...

When coming in over a network connection where a need to SAF authenticate is present, there's a brief period of time between the initial contact and the final authentication, and during that time the thread is on an ID for an "unauthenticated" user. We don't want that thread to do anything bad, so the "unauthenticated userid" has to be very limited in its power. When the ID is authenticated to the true user, then the thread switches to that ID. But for that brief period of time ... it's still a case of "I don't know who you are yet."

This is handled by setting up an ID on z/OS that is under its own segregated group, and has very little power. The sample SAF commands above show a separate group and an unauthenticated guest ID which is RESTRICTED, NOPASSWORD and NOOIDCARD.

This unauthenticated ID needs to have READ to the APPL profile that defines the "security prefix" for the Liberty z/OS server.

Then you add "unauthenticatedUser=" to the <safCredentials> element to define the ID that will be used for this purpose. We're showing 'WSGUEST,' which is the default value for a Liberty z/OS server.

IBM WebSphere Liberty Batch z/OS

SAF Work: Certificate Creation and Keyrings (batchManager client on z/OS)

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty') -
OU('LIBERTY')) WITHLABEL('LibertyCA.LIBERTY') TRUST -
SIZE(2048) NOTAFTER(DATE(2031/12/31))

RACDCERT ID(LIBSERV) GENCERT
SUBJECTSDN(CN('wg31.washington.ibm.com') -
O('IBM') OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY') -
SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY')) SIZE(2048) -
NOTAFTER(DATE(2031/12/31))

RACDCERT ID(SERVERID) ADDRING(Keyring.SERVER)

RACDCERT ID(CLIENTID) ADDRING(Keyring.CLIENT)

RACDCERT CONNECT(ID(SERVERID) -
LABEL('DefaultCert.LIBERTY') RING(Keyring.SERVER)) -
ID(SERVERID)

RACDCERT CONNECT(CERTAUTH LABEL('LibertyCA.LIBERTY') -
RING(Keyring.CLIENT)) ID(CLIENTID)

PERMIT IRR.DIGTCERT.LISTRING -
CLASS(FACILITY) ID(<IDs>) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST -
CLASS(FACILITY) ID(<IDs>) ACCESS(READ)
```

Generate a Certificate Authority
You would not do this if you use a real, well-known Certificate Authority. This is a RACF-generated CA.

Generate the Server Cert and sign with CA
In the real-world you'd generate without a signer, then you'd have a real CA sign it and return along with their public key.

Create a keyring for the server ID
This will be used to hold the server certificate.

Create a keyring for the batchManager ID
This will be used to hold the CA public key.

Connect the server cert to the server keyring
The server will send this certificate to the client.

Connect the CA to the batchManager keyring
The client will use this to verify the server certificate.
Note: if batchManager is off-platform, you would export the CA public key and import it to the client's keyring or TrustStore there

For both IDs, permit keyring access
Access to keyrings may not be UACC(READ)

server.xml ...

Here are the RACF commands to create the profiles in support of SAF keyrings and SAF for SSL. This is a busy chart, so let's walk through the things from top to bottom:

- The first RACF command is to generate a Certificate Authority certificate that will be used to sign the server's certificate. You would not need this if you were having your server certificate signed by a real CA. We show this because it's a way to get a CA-signed server certificate for testing purposes before you've acquired the actual CA-signed certificate from your CA.
- The next block is the generation of the server certificate, with it being signed by the RACF CA we generated initially. This is the certificate that will be sent down to the client.
- Then we create a keyring for the server ID.
- And a keyring for the client ID.
- Then we connect the server certificate to the keyring that's associated with the server ID. When a client seeks an SSL session, the server will go to this keyring for the server certificate and send it down to the client.
- We then connect the CA's public key to the keyring associated with the client ID. This is what allows it to validate the certificate that has come from the server. (In real life you would connect the public key of the CA that signed the server certificate.)
- Finally, we grant the client and server IDs the ability to access and list the keyring to find and retrieve the certificates as needed.

Updates to server.xml to Remove Basic Security and Add SAF for SSL, Registry, Auth

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

<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials unauthenticatedUser="WSGUEST"
  profilePrefix="BBGZDFLT" />

<sslDefault sslRef="DefaultSSLSettings" />
<ssl id="DefaultSSLSettings"
 keyStoreRef="CellDefaultKeyStore"
 trustStoreRef="CellDefaultTrustStore" />
<keyStore id="CellDefaultKeyStore"
 location="safkeyring:///Keyring.SERVER"
 password="password" type="JCERACFKS"
 fileBased="false" readOnly="true" />
<keyStore id="CellDefaultTrustStore"
 location="safkeyring:///Keyring.SERVER"
 password="password" type="JCERACFKS"
 fileBased="false" readOnly="true" />

(all the "basic" security elements removed)
```

Add to the feature list

The ssl-1.0 feature is needed for SAF SSL support.

Add these three lines

The "profilePrefix" value may be any string you want. Showing default here. Key is be consistent with the EJBROLE prefix used (see below).

Note we've specified the unauthenticated user.

Add the SSL Information

Note the keyring. That must match what's been defined for the ID under which the server runs. The password is literally "password" … it's just a dummy string … SAF keyrings don't have passwords.

Remove all the "basic" security elements

To enable the use of SAF keyrings for SSL, we need to do a few things in the server.xml of the server:

- We need to add ssl-1.0 to the feature list. This enables SAF SSL support.
- We need to add the <safCredentials> element with the unauthenticated user specified and the profilePrefix= set to the value we have created with the APPL profile.
- Then we add the block of XML shown in the chart to define the SSL settings. In summary it's defining a set of SSL definitions named "DefaultSSLSettings" and then providing information about the keyrings that represent the key and trust stores. In this example we're showing the same keyring for both (which is common), and it's the keyring we created earlier for the server ID.

  Note: the password value of "password" is *literally* just that. The reason this is there is the SSL settings for Liberty are for all platforms, and on other platforms the SSL keyfile *does* require a password. But on SAF with keyrings no password is required. But the Liberty code still requires *something* there. So we dummy code "password."
- Then all the "basic" security XML is removed.

What About SERVER and EJBROLE?

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

SAF registry and EJBROLE authorization, must have Angel up, and server ID READ to the highlighted SERVER profiles at a minimum

Grant ID of your batchManager to the "allAuthenticatedUsers" EJBROLE

Grant ID of your batchManager user to the profile for access desired

```
RDEF APPL BBGZDFLT OWNER(SYS1) UACC(READ)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.management.security.resource.allAuthenticatedUsers OWNER(SYS1) UACC(NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchAdmin OWNER(SYS1) UACC(NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchSubmitter OWNER(SYS1) UACC(NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchMonitor OWNER(SYS1) UACC(NONE)
```

AdminCenter ...

Once again we return to the SERVER profiles. If we assume the use of just batchManager (thus no WOLA), then the highlighted SERVER profiles are needed. (Recall our conversation about "named Angels" as it relates to the top two SERVER profiles.)

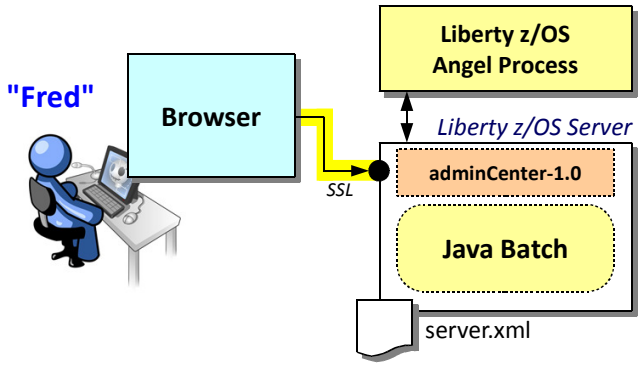
Then in the block of RACF commands at the bottom of the chart:

- The APPL profile defines the "security prefix" that will be used. We grant the unauthenticated user READ to this as well.
- We create the four EJBROLE profiles shown on the chart.
- We grant the batchManager ID READ to the first one -- "allAuthenticatedUsers". This is like a "front door" through which any authenticated user passes. This by itself does not grant the ID any Java Batch authority, however.
- Then we grant the batchManager ID READ to either batchAdmin, batchSubmitter, or batchMonitor, depending on the level of authority we wish to grant that ID.

Client Access

adminCenter (once SAF authentication/authorization/SSL is in place)

Assuming the Setup From the Previous Section … This is Relatively Easy



All the infrastructure to support SAF registry, SAF authorization, and SAF SSL is in place

The difference is that the AdminCenter uses a different EJBROLE from the Java Batch support, so that AdminCenter EJBROLE needs to be created and the ID of the AdminCenter user needs to be given READ to the profile.

```
RDEFINE EJBROLE BBGZDFLT.com.ibm.ws.management.security.resource.Administrator UACC(NONE)

PERMIT BBGZDFLT.com.ibm.ws.management.security.resource.Administrator
  CLASS(EJBROLE) ID(FRED) ACCESS(READ)
```


The access requirements for the AdminCenter are really easy if we assume you've done all the work for batchAdmin. That's because the two share a lot of the same security artifacts. (Both are network-based access mechanisms, so the same issues of SSL and unauthenticated users come into play.

But we do have to add one more thing -- the EJBROLE the AdminCenter employs is different from that used by the Java Batch function. That EJBROLE is shown in the chart. Create that, then grant the ID access. When you point your browser at the server it'll go through the SSL session establishment, then prompt for a userid. The user will supply their SAF ID and password, which will authenticate them. Then it'll check against this new EJBROLE. If they have READ, then they're into the AdminCenter.

Summary

Summary

Security is a big topic

The Java Batch function is mostly just relying on the security support provided by Liberty

Liberty z/OS has SAF as an additional security component

Key things: file security, z/OS authorized service access (Angel), authentication, authorization, and data link encryption

We are to the end of the security unit, and to the end of the workshop presentation material.

Some summary points about security:

- It's a big, big topic. It's very challenging to fit it all -- or even a portion of it all -- into a one-hour presentation.
- Java Batch is really just an "application" within the Liberty runtime, so it's mostly just relying on the security framework Liberty itself is providing.
- On z/OS we have the additional potential of utilizing SAF for security constructs as we've seen.
- The key things are what we discussed -- file security, authorized service access (the Angel process), authentication, authorization, and data link encryption.

End of Unit