



WebSphere Liberty Batch z/OS Java Batch z/OS Security

Version Date: September 6, 2018

See "Document Change History" on page 64 for a description
of the changes in this version of the document

WP102696 at
ibm.com/support/techdocs
© IBM Corporation 2017

This document was written by:

Don Bagwell (dbagwell@us.ibm.com)

David Follis (follis@us.ibm.com)

Scott Kurz (skurz@us.ibm.com)

Table of Contents

Overview	5
Security and batch processing?.....	5
How this document is structured.....	5
Other documents that may be needed.....	6
Important disclaimer about sample security commands.....	6
Liberty Batch Security Concepts Overview	7
Liberty Batch security at a very, very high level.....	7
The three essential elements of security.....	7
Confidentiality.....	7
Authentication.....	7
Authorization.....	8
On z/OS, three more elements of security.....	8
SAF.....	8
Started Task ID.....	8
z/OS Authorized Services.....	9
Basic security vs. SAF security.....	10
Basic security.....	10
SAF security.....	10
Liberty Batch job submission and security ... a road with several forks.....	10
A brief overview of SSL and how it works.....	12
batchManagerZos and WOLA.....	14
What about the AdminCenter?.....	15
Other security considerations related to Liberty Batch.....	15
Access to the DB2 JobRepository.....	16
Access to the Job Logs.....	16
Access to the queueing mechanism in a multi-server environment.....	16
Summary.....	17
Single-Server Environment	18
SAF in support of Liberty server started task.....	18
batchManager access with basic registry and basic role enforcement.....	19
batchManager access with SAF registry and SAF role enforcement.....	22
SERVER profiles in support of SAF authorization (EJBROLE checking).....	22
server.xml updates in support of SAF registry and role checking.....	24
EJBROLE profiles in support of batchManager usage.....	25
batchManager access with SAF keyring for SSL.....	26
SERVER and EJBROLE profiles.....	27
Creation of CA certificate, server certificate, and server keyring with certs.....	27
Updates to server.xml in support of SAF keyring.....	28
Client access to the CA certificate that signed the server certificate.....	28
batchManager access with SAF keyring for SSL and client certificate for authentication.....	30
Generate client certificate.....	30
Connect client certificate to client ID (or export for download to other servers).....	31
Updates to server.xml to support client certificate authentication.....	32
Client-side JVM arguments and batchManager command example.....	32
Differences between batchManager and batchManagerZos.....	33
batchManagerZos access with basic registry and basic role enforcement.....	33
batchManagerZos access with SAF registry and SAF role enforcement.....	34
SERVER profiles in support of WOLA.....	35
server.xml updates in support of WOLA.....	36
CBIND profile in support of WOLA.....	37
EJBROLE profiles in support of batchManagerZos usage.....	38
AdminCenter overview.....	38
AdminCenter access with basic registry and basic role checking.....	40
AdminCenter access with SAF registry and SAF role enforcement.....	41
AdminCenter access with SAF keyring for SAF SSL.....	42
Securing the UNIX file system.....	43
Securing the JobRepository database (DB2).....	43

Identity asserted into DB2.....	44
Access privileges on the DB2 tables.....	44
Authentication alias password encoding.....	45
Multi-Server Environment.....	48
Overview – setting in context to the single server information.....	48
Job submission identity when multi-server is used.....	49
Considerations when security is "basic".....	49
Considerations when security is SAF.....	50
Batch group authorization - 18.0.0.1.....	50
Securing the queueing environment – MQ.....	54
Single-server considerations.....	54
Multi-server.....	55
Multi-server and job partitioning.....	56
Miscellaneous and Reference Information.....	58
Liberty unauthenticated guest.....	58
Summary of updates: batchManagerZos and SAF.....	58
Summary of updates: batchManager with SAF registry and SAF role enforcement.....	60
Summary of updates: batchManager, SAF authentication and roles, SAF SSL.....	61
Document Change History.....	64

Overview

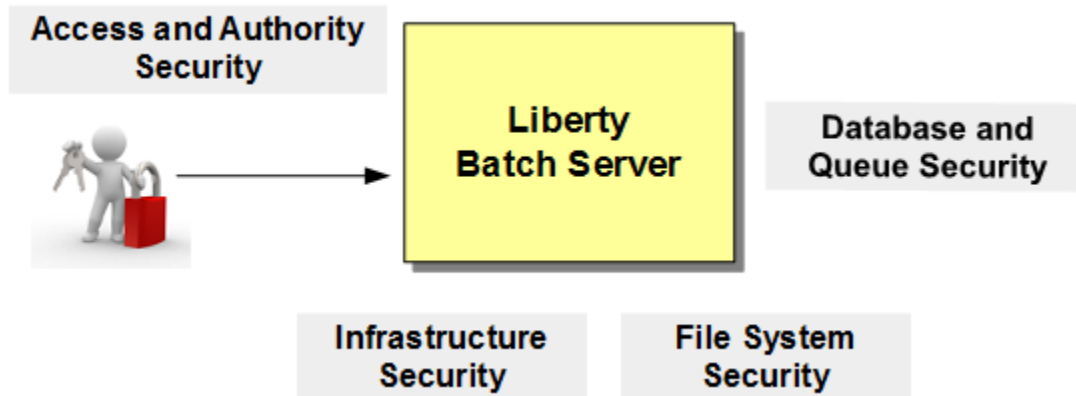
WebSphere Liberty Batch (or "Liberty Batch" for short) is IBM's implementation of the JSR-352 open standard for Java Batch processing. It is available on all operating system platforms supported by Liberty, including z/OS. For more on WebSphere Liberty Batch in general, see:

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102544>

This document will focus on the topic of security within a Liberty Batch runtime environment.

Security and *batch* processing?

Yes, there *are* security considerations to take into account when planning and using Liberty Batch:



The "Access and Authority" security box relates to what users are allowed to submit jobs, view jobs that have been submitted, or purge old jobs. The "Infrastructure" security box relates to things such as SSL (network encryption) and where userids and passwords are maintained. The "Database and Queue" security box relates to Liberty Batch JobRepository database, and the queueing mechanism used when a multi-server configuration is employed. The "File system" security box relates to UNIX permissions on the configuration file system and job log files in the file system.

This can all be made *very simple* when first starting out, or for development and test environments. It can be made *more robust* for production environments.

How this document is structured

This document has four main sections:

Section	Page
"Liberty Batch Security Concepts Overview" A high-level overview of the relevant security concepts related to Liberty Batch is provided. This will provide a framework and context for the details that follow.	7
"Single-Server Environment" In this section we assume a single server topology and we explore the security requirements for the three primary access mechanisms: the AdminCenter (browser); batchManagerZos (command line using WOLA); and batchManager (command line using REST). Under each we will show how to use "basic" definitions (in <code>server.xml</code>), as well as how to move the definitions down to SAF.	18
"Multi-Server Environment" Much of what we covered in the first section applies here as well. Here we go further and explore the security implications of the queueing mechanism between the dispatcher and the executor, be that MQ or the SIBus.	48

<p>"Miscellaneous and Reference Information"</p> <p>We use this section as a repository for information that is important, but does not quite fit elsewhere in the document. Where applicable, we cross-reference to information in this section from elsewhere in the document.</p>	58
--	----

Other documents that may be needed

This document will not attempt to be a "Getting Started" guide for setting up a Liberty Batch environment. The following Techdoc has that:


<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102544>

Specifically, the following two PDF documents included there:

Step-by-Step Implementation Guide

This document is 80+ pages long and provides a fairly detailed step-by-step instructions for configuring and using the WebSphere Liberty Java Batch solution. The document's focus is on the z/OS platform, but a great deal of the configuration information is common across platforms.

This guide can serve as a self-guided "Proof of Technology" for IBM WebSphere Liberty Java Batch.




[WP102544 - WLB Step-by-Step Implementation Guide.pdf](#)

Sample Configuration Illustration

This document provides an illustration of an actual WebSphere Liberty Java Batch configuration we have running in the test lab. This is a multi-server configuration using a "dispatcher" server and two "executor" servers, with IBM MQ as the queuing mechanism between the dispatcher and executors. This also illustrates how batch events operates.

Note: this is a *sample*, and is intended as an illustration only. Please review all security samples and configuration values and modify as needed to comply with your local standards and policies.



[WP102544 - Sample Configuration.pdf](#)

This security-related document assumes Liberty z/OS has been installed, and the essential setup is understood and has been completed. This document will focus more on the security definitions required to accomplish different scenarios. In that sense it will be more of a reference document than a step-by-step guide.

Important disclaimer about sample security commands

Please note the following:

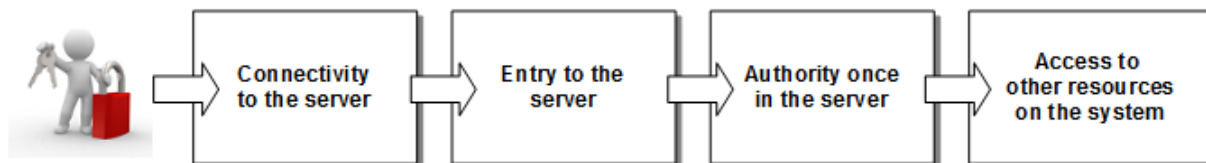
The following documentation includes sample security commands. This sample commands are not part of any standard IBM product and are provided to you solely for the purpose of assisting you in the development of your runtime environment. The samples are provided 'as is', without warranty or condition of any kind. IBM shall not be liable for any damages arising out of your use of the sample commands, even if IBM has been advised of the possibility of such damages.

Every security sample included in this document should be carefully reviewed by your security administrator and implemented only in accordance with your local security policies.

Liberty Batch Security Concepts Overview

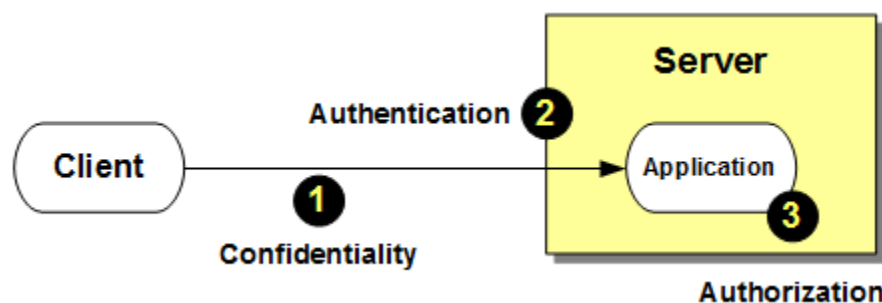
At first it may seem like "batch" would not require much security. But when we look closer, we see there are quite a few places where batch processing requiring protection to prevent unauthorized execution of jobs.

Liberty Batch security at a very, very high level



The security story is really about putting up layers of defense so only those who should get further back in the system can get back. The same is true for Liberty Batch. We will see this basic picture played out time and again as we move through this document.

The three essential elements of security



Confidentiality

This involves *encrypting* any network communications that occur between the client and the server. This is to insure nothing sent or received is seen by those who should not be looking at it. The mechanism for doing this is what most refer to as "SSL," but the more precise term is Transport Layer Security, or "TLS." But "SSL" is so firmly rooted in our language that we'll use that term.

Liberty Batch? The `batchManager` command line interface tool uses REST, which is a TCP/IP network protocol. Encryption of the link between the `batchManager` client and the Liberty server prevents anyone from seeing what's in the communication.

The `batchManagerZos` command line interface tool, on the other hand, uses WOLA, which is a z/OS cross-memory mechanism. No encryption is needed for that since there is no network element involved. It still has security wrapped around it, just not SSL.

To establish encrypted SSL between a client and a server involves the use of *digital certificates*. See "A brief overview of SSL and how it works" on page 12 for more on that topic. Later, when we look at SSL for `batchManager` or the AdminCenter, we'll deal with certificates, key- and trust-stores, and establishing the SSL session.

Authentication

When a client presents itself to a server, the server needs to know who that client is. A client who says, "I'm Fred!" isn't good enough ... the server needs to verify that Fred is who he says he is. That is *authentication*. Typically this is done by having the client present not only its name, but also a password. The server checks the password against a *registry* (a repository of user names and passwords), and if the password given by the client matches what's in the repository, then the server trusts the client is who it says it is.

There are different ways a registry can be implemented, and there are different ways in which a client can assert its identity and be trusted. We'll see those in action later in this document.

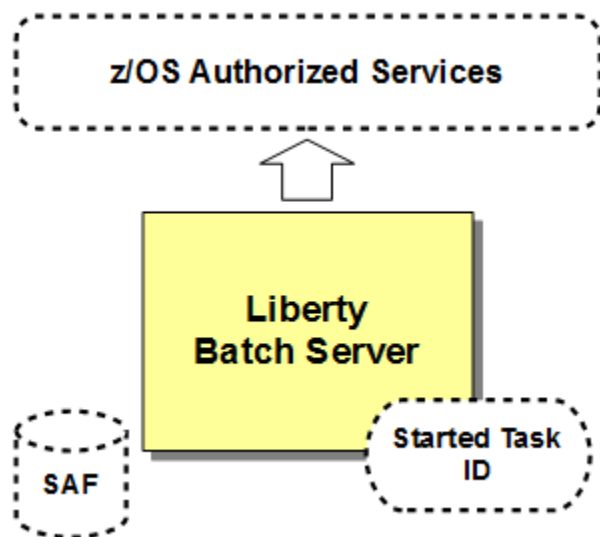
Authorization

Once the client is authenticated, the next question is this: "What is that client allowed to do?" Client "Fred" may be allowed to have full authority to do any administrative role, but client "George" is limited to a smaller set of roles. That is *authorization*. This involves checking the client name against a list of "roles" to see what authority it is to be allowed.

Again, there are different ways these role definitions can be stored so the server can check the authorizations. We'll see those in action later in this document as well.

On z/OS, three more elements of security

The z/OS operating system brings three more things to the table that we must keep in mind:



SAF

SAF stands for "Security Access Facility," and it is a function of the z/OS operating system. It provides an interface through which servers and programs may seek security enforcement using a security program on the other side of the interface, such as IBM's RACF, or a comparable vendor product.

The use of SAF security is very common with z/OS. It is used to store userids and passwords, digital certificates, application authorization roles, as well as protect access to other things on the z/OS system.

Started Task ID

Liberty on z/OS may be started as either a UNIX process – that is, started from a UNIX shell using a supplied shell script – or started as a z/OS Started Task (STC). In either case the Liberty z/OS server is going to operate under a z/OS ID. The question is how the ID it runs under gets assigned.

When Liberty z/OS is started as a UNIX process, the ID it runs under is whatever ID you are logged in as in the UNIX shell environment. For example, if you open a Telnet (or SSH) session to your z/OS system, whatever ID you log in as will be the ID the server runs under when you issue the `server start` command¹.

¹ Well, unless you `su` to another ID before you issue the `server start` command. But the main point still applies – the ID in effect when you issue the `server start` command is the ID under which the Liberty server will operate.

We do not anticipate many will start their Liberty Batch servers as UNIX processes. Perhaps at first during initial testing, but certainly not later when the servers are part of a more formalized test or production environment.

Most will start their Liberty z/OS servers as "started tasks," which involves issuing a z/OS START command, which involves a "JCL Start Procedure" to define how the started task is to operate.

On z/OS, the ID under which a started task operates is determined with a SAF `STARTED` profile. The `STARTED` profile defines what ID is to be assigned based on the JCL start procedure used². In effect: "If JCL name is `BATCHSRV`, then assign ID of `BATCHID`."

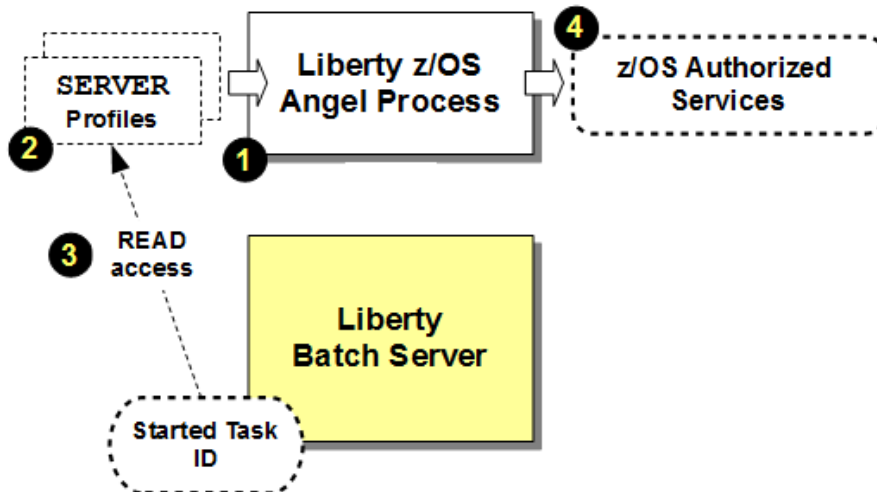
Before you can assign an ID to a started task, that ID must exist. So a necessary first step is to define the ID. Then you define the `STARTED` profile. Then you're ready to start your server as a started task.

With Liberty Batch on z/OS, if we assume the server is started as a started task, then creating the ID and `STARTED` profile is necessary, *regardless* of what you do with the Liberty server later. We cover this in more detail under "SAF in support of Liberty server started task" starting on page 18.

z/OS Authorized Services

In the z/OS operating system there are some things most IDs are free to do, other things that are a bit more restricted, and certain things that are strictly controlled. The strictly controlled functions are known as "authorized services" – one must be *authorized* to access and use them.

This where the Liberty z/OS "Angel Process" comes into the picture. It is the running Angel process, and SAF `SERVER` profile definitions, that protect access to z/OS authorized services:



Notes:

1. If the Angel Process is running, *and*
2. The appropriate SAF `SERVER` profiles are created, *and*
3. The server started task ID has `READ` to the SAF `SERVER` profiles, *then*
4. The Angel Process will allow access to the z/OS authorized services.

What "authorized services" does Liberty Batch require? Well, it depends on what you're doing. For example, the `batchManagerZos` command line client will require access to

² Or the z/OS `JOBNAME` as well, but for now let's stay focused on the ID being assigned based on the JCL start procedure name.

authorized services because it uses WOLA³, and WOLA is an authorized service. The `batchManager` command line client may not, depending on whether you're using SAF for authorization role checking⁴.

Basic security vs. SAF security

In the next section we will provide a picture that illustrates the choices you have for submitting batch jobs, and in that picture we will make reference to "basic security" and "SAF security." Here's what we mean by each:

Basic security

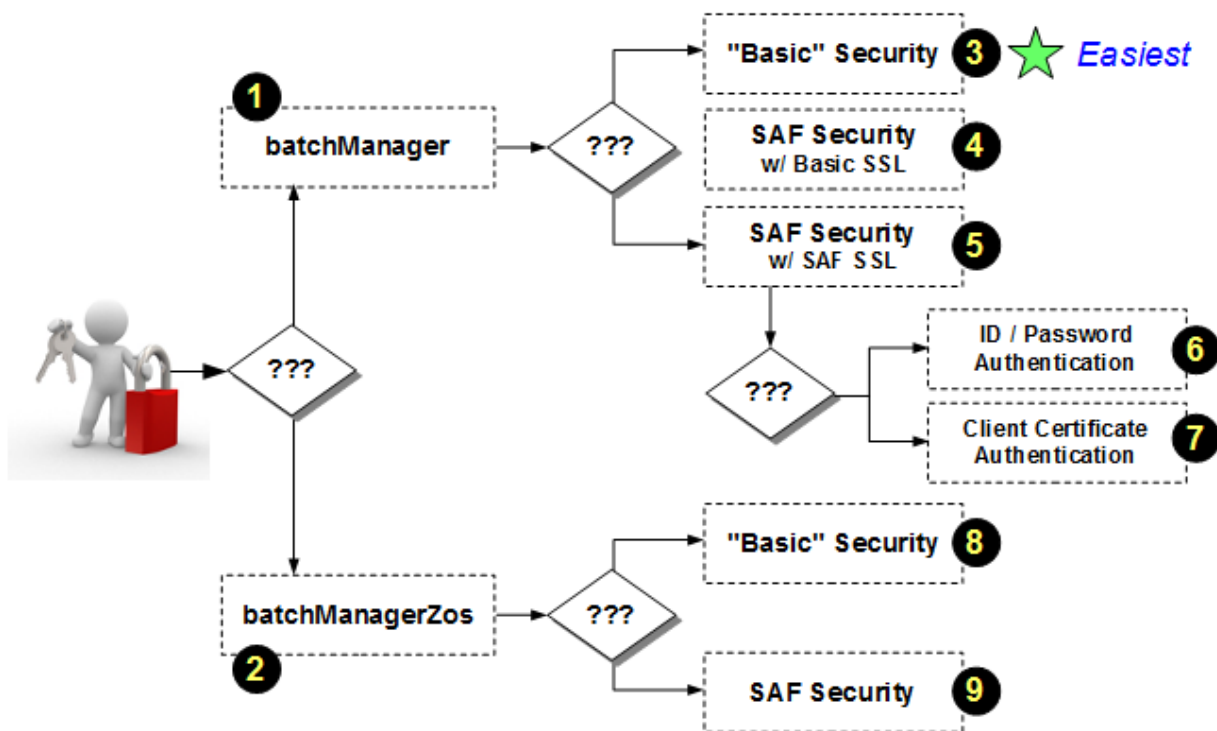
By "basic" we mean the Liberty server is providing the security enforcement. The definitions for things such as the user registry and authorization roles are spelled out in the `server.xml` configuration file. The setup of this security is simple because you do not need to involve anyone else; if you have access to the `server.xml` file, you can set up this security.

SAF security

In this case we mean the security definitions are held in SAF. The `server.xml` configuration file has entries that point to SAF, but things like the userids and password, or the application role definitions, are not in the XML. They are in SAF.

Liberty Batch job submission and security ... a road with several forks

The following picture illustrates the forks in the road. The numbered circles correspond to the notes that follow:



The first fork in the road is the decision about which command line utility you will use to submit your batch jobs.

³ WebSphere Optimized Local Adapters, a cross-memory message exchange mechanism.

⁴ EJBROLE checking in SAF is an authorized service. Using SAF as a user registry does not require access to authorized services.

❶ batchManager

This is a command line utility that uses REST, which is a network protocol based on TCP and HTTP. That means there's a requirement for SSL. But that can be simplified by using "basic" security (block ❸). The batchManager client does not require the Angel or access to authorized services unless you use SAF for role checking (block ❹).

❷ batchManagerZos

This is a command line utility that uses WOLA (cross-memory) to communicate with the Liberty Batch server. Because WOLA is cross-memory, there is no network involved. That means there's no SSL needed. WOLA is an authorized service, which means the Angel Process is needed, along with the appropriate SAF SERVER profiles to permit access to those authorized services.

With the batchManager command line client you have three options for security:

❸ "Basic" Security

The term "basic" means the security definitions are in the `server.xml` configuration file of the server. It is adequate for development and testing, but would not be acceptable for production.

Unlike basic security and batchManagerZos, basic security with batchManager works well. As the picture notes, this is the easiest approach when first starting out.

Basic security requires no access to authorized services, so you can do this without the Angel Process and without any SAF SERVER profiles. That's why this is the easiest.

We cover this option in detail under "batchManager access with basic registry and basic role enforcement" starting on page 19.

❹ SAF Security w/ Basic SSL

SAF – "Security Access Facility" – is an interface provided by z/OS behind which resides a security product such as IBM RACF, or a comparable vendor product. SAF can be used to hold the user registry as well as enforce authorization roles.

SAF role checking (using EJBROLE profiles) is an authorized service, and that means the Angel Process is required as well as the appropriate SAF SERVER profiles, and the server ID granted READ to those profiles.

"Basic SSL" means the server certificate used for SSL is auto-generated by Liberty. It's adequate for development or testing, but not for production. We offer this because it's a stepping-stone⁵ to "full SAF" (block ❺). See "A brief overview of SSL and how it works" on page 12 if you're unfamiliar with SSL and certificates.

We cover this option in detail under "batchManager access with SAF registry and SAF role enforcement" starting on page 22.

❺ SAF Security w/ SAF SSL

This is the "full SAF" option, with the user registry (authentication), the role checking (authorization), and SSL (encryption) all handled in SAF.

This is really just the previous option (block ❹) with the Liberty auto-generated certificate information removed from the `server.xml` configuration file, and the SAF SSL information added. It also involves creating the server certificate in SAF that is signed with a certificate authority. See "A brief overview of SSL and how it works" on page 12 if you're unfamiliar with SSL and certificates.

We cover this option in detail under "batchManager access with SAF keyring for SSL " starting on page 26.

Just when you think we could not possibly have any more options, we add two more related to the last option (block ❺) – authenticating with a passed-in userid and password, or authenticating with a "client certificate" that represents a SAF ID.

⁵ You can get SAF working for the user registry and role authorization without having to worry about creating the server certificate in SAF and signing it with a certificate authority. Then when you get SAF registry and authorization working, you can work on SAF SSL.

6	<p>Authentication with passed-in userid and password</p> <p>The batchManager command line client provides the ability to pass in the userid and password for the user to be authenticated. It is done with the <code>--user=<id></code> and <code>--password=<password></code> options on the command line.</p> <p>This option is part of the discussion of option 4 above.</p>
7	<p>Authentication with a client certificate</p> <p>Rather than pass in the userid and password on the command line, you can instruct the Liberty server to request of the client a "client certificate" that represents a SAF ID. The advantage of this is it avoids coding a password on the command line.</p> <p>We cover this in detail under "batchManager access with SAF keyring for SSL and client certificate for authentication" on page 30.</p>

With the batchManagerZos command line client you have two options for security:

8 "Basic" Security

The term "basic" means the security definitions are in the `server.xml` configuration file of the server. It is adequate for development and testing, but would not be acceptable for production. Basic security with batchManagerZos *is* possible, but it has a quirk in that the ID that is asserted over the WOLA connection is not accessible to basic authentication and authorization. You can work around that by turning off security. Again, acceptable for development or test, but not production.

Because WOLA is used – an authorized service-- it means the Angel Process is needed along with the appropriate SAF `SERVER` profiles, and the server ID granted `READ` to those profiles.

We cover this option in some detail under "batchManagerZos access with basic registry and basic role enforcement" starting on page 33.

9 SAF Security

SAF – "Security Access Facility" – is an interface provided by z/OS behind which resides a security product such as IBM RACF, or a comparable vendor product. SAF can be used to hold the user registry as well as enforce authorization roles.

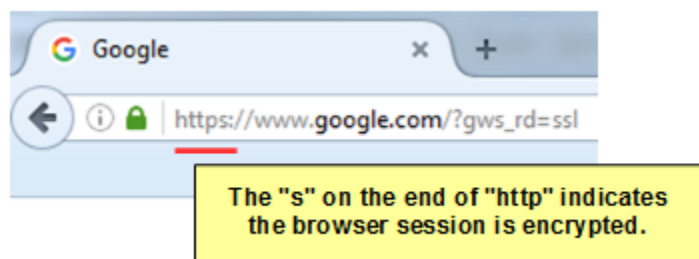
This is the more likely path when using batchManagerZos.

Because WOLA is used – an authorized service-- it means the Angel Process is needed along with the appropriate SAF `SERVER` profiles, and the server ID granted `READ` to those profiles. Using SAF for authorization role checking is also an authorized service which requires the Angel Process and a few other SAF `SERVER` profiles beyond what's needed for WOLA.

We cover this option in detail under "batchManagerZos access with SAF registry and SAF role enforcement" starting on page 34.

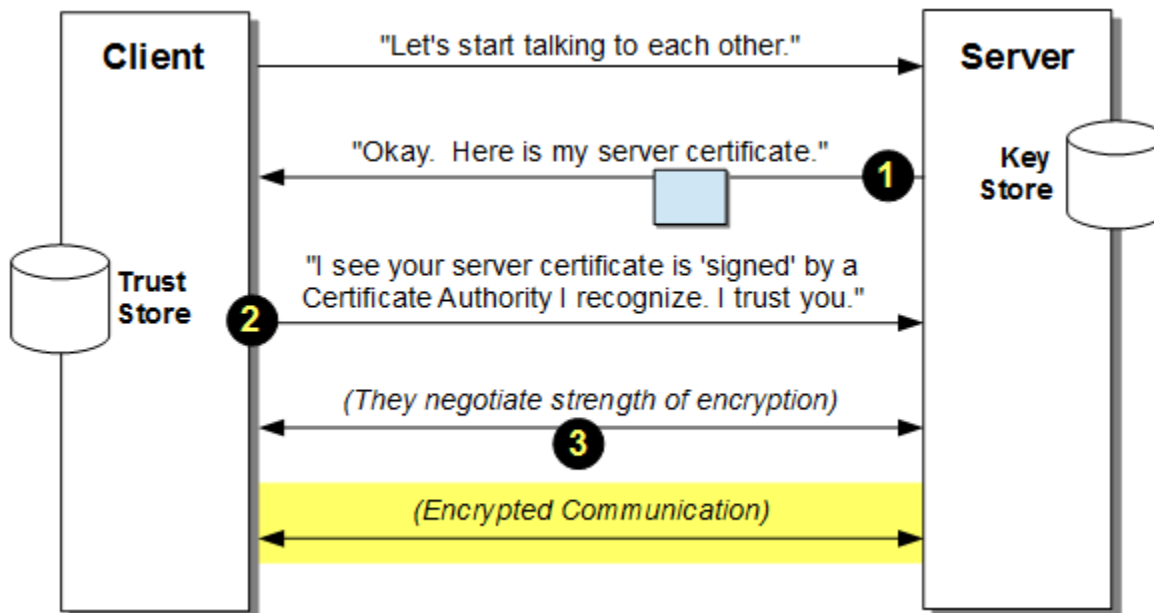
A brief overview of SSL and how it works

The purpose of SSL⁶ is to encrypt communications between the client and the server so anyone "sniffing" the network will not be able to read the messages. You use SSL every day when you use a browser on the Internet:



⁶ As mentioned earlier, SSL is a still-used-but-outdated term. The more correct term is TLS (Transport Layer Security).

To establish this connection, the client and the server go through a "handshake" to establish a number of things before building the SSL connection. The following picture illustrates this and provides a simplified overview:

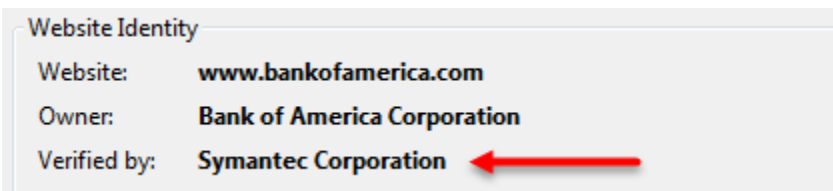


The client starts by sending a request into the server to begin talking to each other. When a browser is the client, that is done by typing and entering a URL. Often that will first go to the non-SSL port (default port 80) and be *redirected* to the SSL port (default 443).

Then the server sends to the client its "server certificate," (❶) which is a way for the client to be able to trust it is talking to the server it thinks it's talking to. This "server certificate" is really just a string of jumbled characters (that is, encrypted and appearing as random mixed characters). The server keeps its server certificate in a "key store" that is accessible to the server.

Note: This "key store" is typically a file in a file system, but on z/OS it can also be a "keyring" in SAF. The use of keyrings to hold digital certificates is something we explore later in this guide.

In order for the client to trust the server is who it says it is, the server certificate is "signed" by what's called a "Certificate Authority," or CA for short. A CA is an organization that vouches for the authenticity of certificates. For example, the Bank of America website shows this:



That means the Bank of America server certificate has been "signed" by Symantec⁷. Users of that site trust the server certificate because we trust Symantec: "If Symantec signed it, then I trust it is valid."

Note: The details of "signing" and "public" keys is part of the "asymmetric cryptography" topic. The specifics of that are well beyond the scope of this document. If you're interested, see: https://en.wikipedia.org/wiki/Public-key_cryptography

Going back to our illustration above, the numbered circle ❷ represents the process of the client inspecting the server certificate it received using the public key of the CA that signed the

⁷ This implies no endorsement of either Bank of America or Symantec. We use this simply as an example.

certificate. The client keeps CA public keys in its "trust store," which may be a file in the file system, or on z/OS it may be a SAF keyring.

If the browser client does not recognize the CA that signed the server certificate – that is, it does not have that CA's public key in its trust store – it will issue a warning indicating the server may not be trustworthy. Perhaps you have seen that when using a browser on the Internet. Browsers come with a long list of "well-known" CA public keys that are part of its trust store when you install the browser. A CA that is *not* well-known will result in the browser asking you whether you want to trust the site. If you answer "yes," you may add that CA public key to your trust store, and that site will be trusted thereafter.

We mention this because the examples provided in this document are for a "self-signed" certificate, which means we – RACF in our case – served as both creator of the server certificate *and* signer of that certificate. We have you export from RACF the CA public key and make it available to the batchManager client trust store. That allows the SSL connection to be established properly.

Once the server certificate is trusted, the client and the server then determine the highest level of encryption that is supported by both of them. That "negotiation" (block ③) goes back and forth a time or two before they settle on an encryption strength⁸. Then the SSL connection is built and from that point on communications flow over the encrypted session.

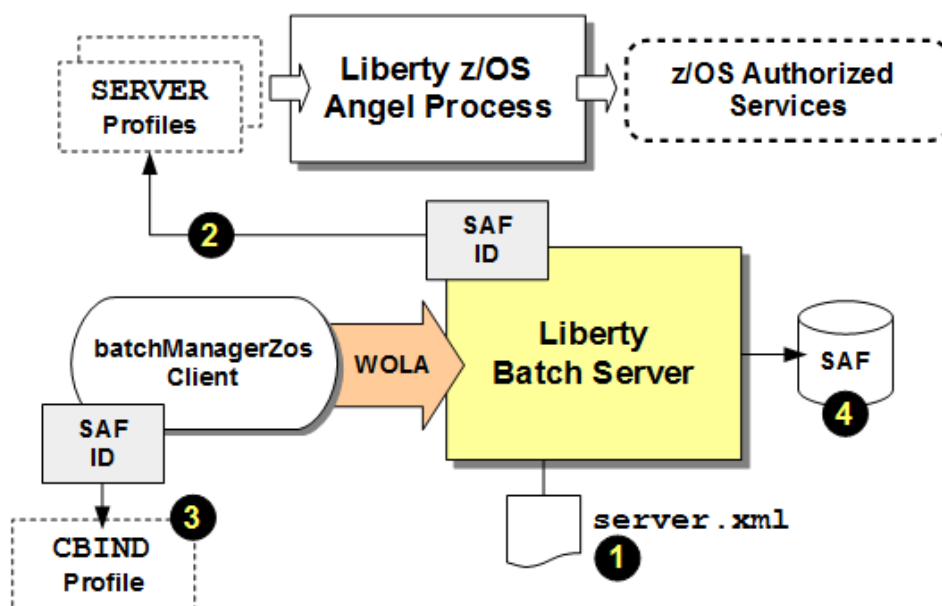
In summary:

- The server has a digital certificate that is used to represent the server. It keeps that digital certificate in a "key store," which may be a file or a z/OS SAF keyring.
- That server certificate is "signed" by a Certificate Authority (CA), which allows the client to determine if the server certificate can be trusted.
- The client must have access to the CA's public key to determine if the server certificate signed by that CA can be trusted. The CA's public key is kept in a "trust store" accessible to the client. That may be a file or a z/OS SAF keyring.
- When the client trusts the server, the two negotiate on an encryption strength to use.
- The SSL session is then established and the communications are encrypted before they flow over the network.
- SSL is applicable to the batchManager command line utility, but does *not* apply to the batchManagerZos command line utility. That's because batchManagerZos uses WOLA, which is a cross-memory communication mechanism. It does not use the network; it is cross-memory. Therefore, since no network is used no network encryption is necessary. It is virtually impossible to "sniff" a cross-memory communication on z/OS.

batchManagerZos and WOLA

The last bullet in the previous section said SSL does not apply when batchManagerZos is used. Does that mean batchManagerZos has no security associated with it? No, it has a fair amount of security associated with it. Take a look at the following picture and follow the number circles with the notes below:

⁸ The details of this are not important here; the main point is the two sides have to agree on the encryption to be used.

**Notes:**

1. You must configure a few things into the `server.xml` configuration file before WOLA can be used at all. Absent those few things, someone trying to sneak in with WOLA will fail because the server itself is not configured to use WOLA.
2. WOLA is an authorized service, which means the Liberty z/OS server is only going to have access to that authorized service if (a) the Angel is running, (b) the WOLA SAF `SERVER` profiles have been created, and (c) the server's started task SAF ID has been granted READ to those profiles. Absent that, WOLA won't work with the server.
3. Let's say 1 and 2 are in place. What's to prevent a user running `batchManagerZos` from coming in? The SAF `CBIND` profile, which allows only those SAF IDs with read to the profile from being able to create a WOLA "registration" into the Liberty server. If the user does not have READ to the `CBIND` profile, they are rejected.
4. Let's say 1, 2 and 3 are in place ... the ID trying to use `batchManagerZos` will be propagated across WOLA into the server, and it still faces *authorization*, just like someone using `batchManagerZos` faces. Only if the ID trying to use `batchManagerZos` has been authorized to the application role (a SAF `EJBROLE`) may they submit a batch job.

All this is covered in some detail under "batchManagerZos access with SAF registry and SAF role enforcement" starting on page 34. The point here is that even though WOLA does not use SSL, it *still* has a lot of security associated with it.

What about the AdminCenter?

The AdminCenter is a browser-based administrative tool that can be used to monitor batch jobs. Because it is browser based – and thus network/HTTP based – the security considerations are similar to what we saw for `batchManager`: that is, *encryption* between the browser and the server; *authentication* to be allowed into the server; and *authorization* to be allowed access to the application.

Like `batchManager`, the security can be "basic" (defined in `server.xml`) or in SAF.

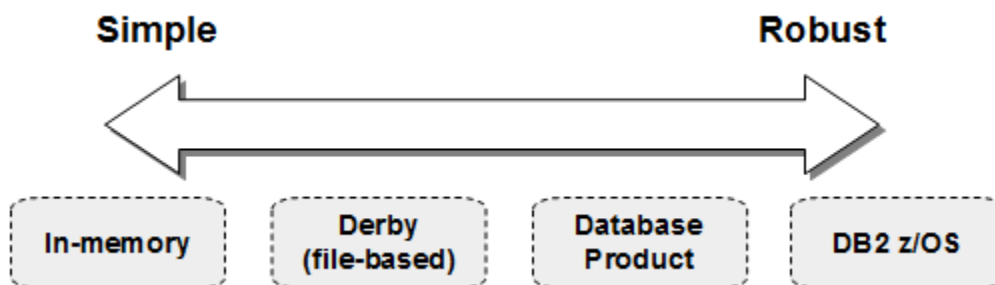
We cover this in some detail starting on page 40 (basic security) and 41 (SAF security).

Other security considerations related to Liberty Batch

In addition to the things discussed above, we have several other aspects of security to take into account as the Liberty Batch environment becomes more sophisticated:

Access to the DB2 JobRepository

The "JobRepository" is a relational database table structure under Liberty Batch that is used to keep track of submitted jobs and their state. Like most things related to Liberty Batch, it can be made simple, or it can be made more robust:



The focus in this document will be on the "robust" side of the spectrum: when the JobRepository is implemented in DB2 z/OS. The focus here is on how to secure the DB2 z/OS tables to allow only authorized access to them.

Access to the Job Logs

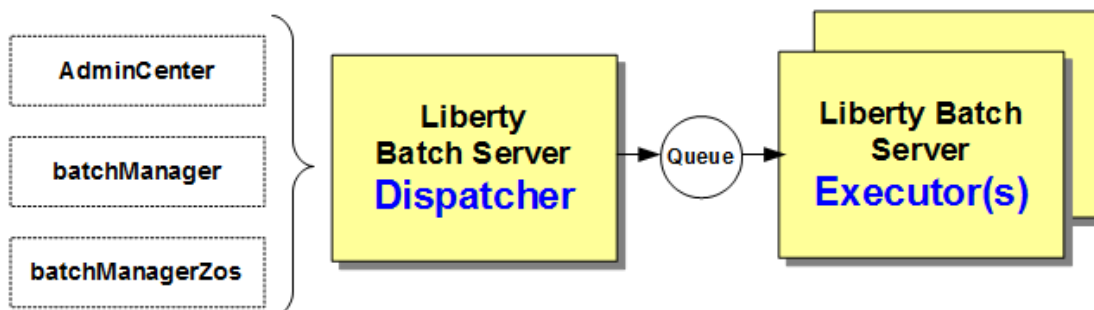
The Liberty Batch job logs are written to a file system location. There are two security considerations here:

1. UNIX file permissions on the job log directories and files so only people authorized to access those files can do so, and
2. Liberty Batch administrative access to the job logs for the purpose of viewing them or purging them.

The first involves standard UNIX file permissions management. The second is really a matter of what authority a user has when they come through the AdminCenter, batchManager or batchManagerZos interfaces. We cover both UNIX file permissions and administrative authority later in this document.

Access to the queueing mechanism in a multi-server environment

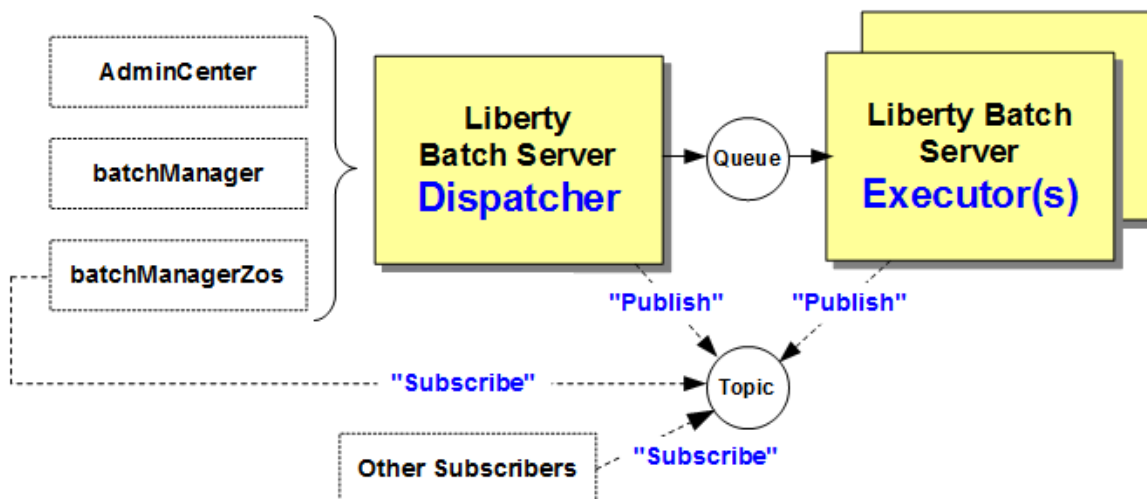
The "queueing mechanism" comes into the picture when we move from a single server environment to a multi-server environment. In a multi-server environment the job submission role is split into "Dispatcher" and "Executor." The Dispatcher hosts the administrative interface (AdminCenter, batchManager, or batchManagerZos); and the Executor is where the batch jobs run:



The "Queue" circle between the two represents either the built-in messaging function of Liberty or IBM MQ.

The "multi-server" topology provides considerable flexibility in how and where batch jobs are executed. From a security point of view, the introduction of the queue in the middle of the picture means access to that queue needs to be properly restricted, and it means we have to take into account the assertion of the authenticated ID over to the Executor servers.

This picture needs to be enhanced a bit because there's another "queueing" aspect to this – the publishing of "batch events" to a pub/sub topic:



Note: The "batch events" function is not limited to the multi-server environment; it may also be employed with a single server. We introduce it here simply because the multi-server environment brought queueing to the discussion, and the pub/sub batch events subject naturally follows.

The security consideration here is two-fold: who can publish to the topic space, and who can subscribe to it.

Summary

The topic of Liberty Batch security has many layers, from relatively straight-forward elements such as *authentication* (challenging the user to prove they are who they say they are), down to more z/OS-specific security. It can be summarized in a few pages such as we've done in this section, but to cover it in full detail requires ... well ... more detail.

The remainder of this document covers the specifics of this in more detail.

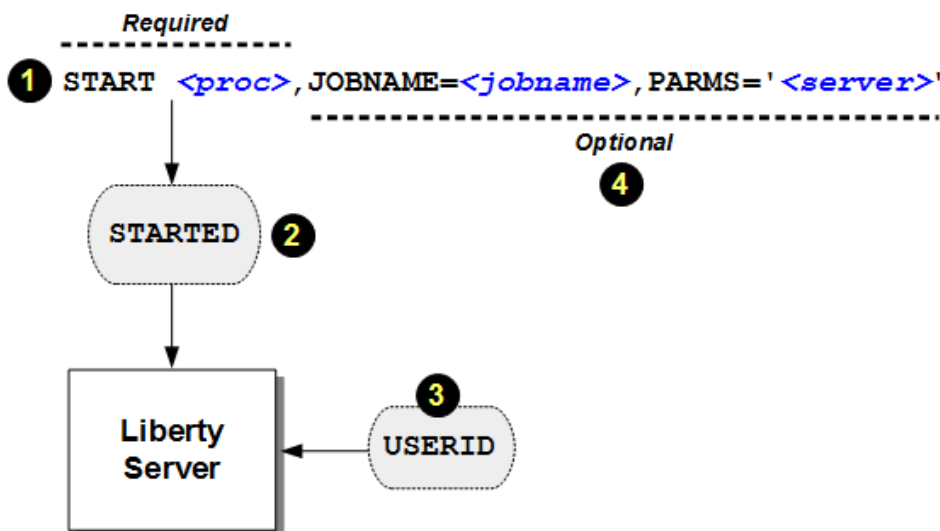
Single-Server Environment

SAF in support of Liberty server started task

For a Liberty server to start as a z/OS started task (STC), there must be an ID and group to assign to the STC, and a `STARTED` profile that assigns the ID based on the `START` command.

Note: This is common to any started task, not just Liberty and WebSphere Liberty Batch.

In picture format:



Notes:

1. The MVS `START` command is issued. The key with respect to the `STARTED` profile is the JCL procedure name, though the optional `JOBNAME=` value may also play a role depending on how the `STARTED` profile is defined.
2. The SAF `STARTED` profile defines the mapping of the `START` command to an ID.
3. The ID is assigned to the started task.
4. Optionally, a `JOBNAME=` value can be provided to make ID assignment more granular than just the JCL proc name. Also, for Liberty z/OS, the server name can be passed in on the `START` command.

Note: Or the server name can be coded in the `PARMS=` statement of the first line of the Liberty z/OS JCL start proc and the `PARMS=` on the `START` command can be omitted.

This picture illustrates what the sample commands that follow will create:



Sample RACF® commands to accomplish the above are provided here:

⁹ Here we illustrate IBM's SAF security product RACF; other SAF security products may be used as well. The syntax for those other products may be different.


```

ADDGROUP <angel-group> OMVS (AUTOGID) OWNER (SYS1)
ADDUSER <angel-user> DFLTGRP (<angel-group>) OMVS (AUTOUID HOME (/ <path>)
PROGRAM (/bin/sh)) NAME ('LIBERTY ANGEL') NOPASSWORD NOOIDCARD
ADDGROUP <server-group> OMVS (AUTOGID) OWNER (SYS1)
ADDUSER <server-user> DFLTGRP (<server-group>) OMVS (AUTOUID HOME (/ <path>)
PROGRAM (/bin/sh)) NAME ('LIBERTY SERVER')
ALTUSER <server-user> PASSWORD (<password>) NOEXPIRED
RDEFINE STARTED <angel-proc>.* UACC (NONE) STDATA (USER (<angel-user>)
GROUP (<angel-group>) PRIVILEGED (NO) TRUSTED (NO) TRACE (YES))
RDEFINE STARTED <server-proc>.* UACC (NONE) STDATA (USER (<server-user>)
GROUP (<server-group>) PRIVILEGED (NO) TRUSTED (NO) TRACE (YES))
SETROPTS RACLIST (STARTED) REFRESH

```

Notes:

- The "Angel process" is used to control access to z/OS authorized services. You will need this if you intend to use WOLA (that is, the batchManagerZos command line utility), or use SAF for role enforcement, or use JDBC Type 2 to access DB2. You may wish to consider creating the Angel ahead of time in anticipation of using it later.

The Angel group and ID should be separate from any other groups and IDs.

The Angel ID should be one that can *not* be used to log into UNIX shell.

- The use of `AUTOGID` and `AUTOUID` is based on your local security practices. If you choose to specify a GID and UID, then modify the syntax accordingly.
- The ID assigned to the server does *not* require a password for Liberty to operate. That is shown here simply because it is convenient to open a UNIX shell to the system and log in with that ID when creating the server initially. Once the server is created, the ID can be set to have no password.

The default JCL proc name for a Liberty z/OS server is `BBGZSRV`, so if that was the JCL proc name used, then the `STARTED` profile would be:

```
RDEFINE STARTED BBGZSRV.*
```

If your server start command was `S BBGZSRV, JOBNAME=ABC`, and you wanted to assign a specific ID based on the jobname, the `STARTED` profile would be:

```
RDEFINE STARTED BBGZSRV.ABC
```

This is fairly standard `STARTED` profile processing. There's nothing special about Liberty z/OS or Liberty Batch when it comes to this.

batchManager access with basic registry and basic role enforcement

In this scenario all the security elements are configured in the `server.xml` file; there is no SAF involvement for SSL, authentication or authorization.

Note: You still need a SAF ID for the server and SAF `STARTED` profiles to assign the ID when you start Liberty as a started task. See "SAF in support of Liberty server started task" on page 18. For this scenario the Angel is not needed, though you may have it running if you wish.

The following XML¹⁰ provides the ID of "Fred" (and only Fred) access as an administrator:

```

<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>

```

¹⁰ This is not the complete `server.xml`, some elements have been omitted to maintain focus on the relevant XML. For example, the JDBC definitions for the job repository are not shown here but were present in the server we used for testing. Also, the HTTP definitions are not shown here but were present in our test server.

```

<feature>appSecurity-2.0</feature>
</featureManager>

<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">
    <user name="Fred" />
  </security-role>
</authorization-roles>

```

Notes:

- The appSecurity-2.0 feature is what turns on authentication and authorization.
- The <keyStore> element is what provides a very basic self-signed SSL certificate that is just enough to allow the establishment of SSL between the client and the Liberty server.

Note: To make this work, you have to give your batchManager UNIX environment access to the key.jks file that is generated by Liberty. The easiest way to do this is to copy the key.jks file to some location accessible by the ID that runs batchManager, and exporting a JVM argument:

```
export JVM_ARGS="-Djavax.net.ssl.trustStore=/<path>/key.jks"
```

The WP102544 Techdoc "Quick Start Guide" provides detailed step-by-step instructions on locating that file, copying it, and setting the JVM_ARGS variable.

- The <basicRegistry> section provides the user ID name and password pairs that will be used for authentication. In this example, only "Fred" is defined.

Note: The basic registry function is case-sensitive for both the name and password values.

- The <authorization-roles> and <security-role> elements provide the framework for authorizing users to batch roles. In this example we are defining just one role – batchAdmin, and we're granting just "Fred" access to it.
- Therefore, in this example, only an ID of Fred and password fredpwd will be accepted for authentication. Once authenticated, Fred will assume the batch administrator role because his ID is defined to the batchAdmin role. An ID of Mary would be rejected with an HTTP 401 error because "Mary" is not defined in the basic registry. In fact, any ID *other than* Fred will be rejected with a 401 based on this set of definitions.

The following XML shows how additional users can be added to the registry, and all three batch roles (batchAdmin, batchSubmitter, and batchMonitor) can be defined:

```

<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
</featureManager>

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

<basicRegistry id="basic1" realm="jbatch">
  <user name="Fred" password="fredpwd" />
  <user name="Mary" password="marypwd" />
  <user name="Bob" password="bobpwd" />
</basicRegistry>

```

```

<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <user name="Fred" />
  </security-role>
  <security-role name="batchSubmitter">
    <user name="Mary" />
  </security-role>
  <security-role name="batchMonitor">
    <user name="Bob" />
  </security-role>
</authorization-roles>

```

Notes:

- In this example Fred, Mary, or Bob can be authenticated, but nobody else can.
- Each is assigned to their respective role.
- Bob has the most limited role with batchMonitor. If he tried to *submit* a job he would encounter the following message:

```
Error 401: CWWKY0304W: User Bob is not authorized to start batch jobs.
```

That's an example of the role-checking function doing its job: Bob has "monitor," not "submitter," so he is rejected in his attempt to submit a job.

Suppose you wanted to make the role of monitor open to any user who is properly authenticated. You can accomplish that with the following:

```

<basicRegistry id="basic1" realm="jbatch">
  <user name="Fred" password="fredpwd" />
  <user name="Mary" password="marypwd" />
  <user name="Bob" password="bobpwd" />
  <user name="David" password="davidpwd" />
  <user name="Scott" password="scottpwd" />
  <user name="Don" password="donpwd" />
</basicRegistry>

<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <user name="Fred" />
  </security-role>
  <security-role name="batchSubmitter">
    <user name="Mary" />
  </security-role>
  <security-role name="batchMonitor">
    <special-subject type="ALL_AUTHENTICATED_USERS" />
  </security-role>
</authorization-roles>

```

Notes:

- In this example David, Scott and Don have no *explicit* assignment to a role.
- But because ALL_AUTHENTICATED_USERS is specified as a "special-subject" under the batchMonitor role, they would have monitor rights once they successfully authenticated.
- A user not named in the basic registry – for example, "Roger" – would be turned away with an 401 authentication failure error message.

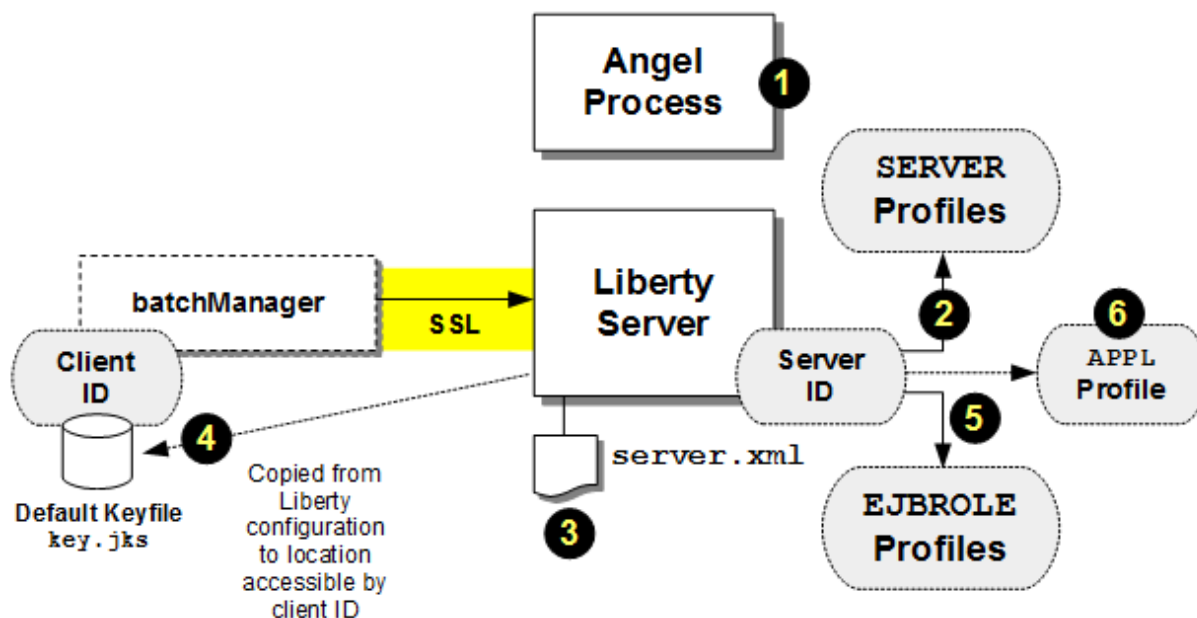
That was "basic" registry and "basic" role definitions. The next step is to push those function down into SAF.

batchManager access with SAF registry and SAF role enforcement

In this section we will explore what's required to have authentication and role checking performed by SAF when the batchManager command line client is used.

Note: For now we are going to illustrate use of the default Liberty keystore for SSL. In the next section we'll show to move that into SAF as well.

The following picture illustrates the needed security components:

**Notes:**

1. To perform SAF EJBROLE checking, the Angel process must be started.
2. The ID (or group) assigned to the Liberty Server needs to have READ to a set of SAF `SERVER` profiles that grant access to the Angel and to the SAFCREATED authorized services
3. The `server.xml` file needs to have a few definitions added indicate SAF is to be used for authentication and authorization.
4. To establish the SSL session, the batchManager client requires access to a keystore that contains the signer certificate that is used by the Liberty server.

Note: In this section we illustrate the use of the default Liberty key file (`key.jks`), but in the next section ("batchManager access with SAF keyring for SSL " on page 26) we build on this section by moving the certificates into SAF keyrings.

5. The ID you supplied on the batchManager command is checked against `EJBROLE` profiles to establish whether it has the authority to use the WebSphere Java Batch function.
6. If your APPL class is active, then an APPL profile matching the security prefix used on the EJBROLE profiles will be needed.

We'll illustrate each with the samples that follow¹¹.

SERVER profiles in support of SAF authorization (EJBROLE checking)

Here's a list of the `SERVER` profiles related to Liberty z/OS. Not all of these are required to use EJBROLE checking (we will explain what's need in a bit); we are showing you the full list to give you a sense for what the `SERVER` profiles are used for.

¹¹ For a summary, see "Summary of updates: batchManager with SAF registry and SAF role enforcement" on page 60.

BBG.ANGEL
BBG.AUTHMOD.BBGZSAFM

Controls access to the Angel process and the authorized services protected by the Angel process

BBG.AUTHMOD.BBGZSAFM.LOCALCOM
BBG.AUTHMOD.BBGZSAFM.WOLA
BBG.AUTHMOD.BBGZSCFM
BBG.AUTHMOD.BBGZSCFM.WOLA

Controls access to the WOLA function. Note the subtle difference in the third qualifier name.

BBG.AUTHMOD.BBGZSAFM.SAFCRED
BBG.SECPFX.BBGZDFLT

Controls access to SAF for certain security functions, most notably EJBROLE checking.

BBG.AUTHMOD.BBGZSAFM.TXRRS
BBG.AUTHMOD.BBGZSAFM.ZOSDUMP
BBG.AUTHMOD.BBGZSAFM.ZOSWLM
BBG.AUTHMOD.BBGZSAFM.ZOSAIO

Controls access to other functions – RR S, DUMP, WLM, and Asynch I/O.

There are two steps to this – (1) creating the SERVER profiles; and (2) granting READ to the ID (or group) you wish to have access to the authorized services.

To create the profiles, the following commands illustrate the syntax:

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE) OWNER(SYS1)
```

Note: The UACC(NONE) means the "universal access" is "none" – that is, by default nobody can access. So the *existence* of a profile does *not* grant the access. Access is granted with a later PERMIT command that grants and ID or group the READ access.

You may wish to consider creating *all* the profiles even though you are only using *some* for WOLA. The presence of those yet unused profiles with UACC(NONE) would cause no harm.

To grant an ID (or group) the ability to perform EJBROLE checking would imply the following commands:

```
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(<id>)
```

The value BBGZDFLT is highlighted because that is a configurable value. The string BBGZDFLT is the default value for the "security prefix," though you may choose to use a

different string. The value here relates to the prefix used on the `EJBROLE` profiles, which are examined a bit later.

Finally, a refresh is needed:

```
SETOPTS RACLIST(SERVER) REFRESH
```

server.xml updates in support of SAF registry and role checking

The following `server.xml` elements are needed to support use of `batchManager` with SAF authentication and SAF role checking.

In the `<featureManager>` section of the file:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
</featureManager>
```

Notes:

- The `batch-1.0` and `batchManagement-1.0` elements are needed to implement the Java batch function.
- The `appSecurity-2.0` feature is what enables role checking.
- The `zosSecurity-1.0` feature is what enables SAF as the registry and role repository.
- Other features may be present; these are the essential features related to SAF authentication and authorization for `batchManager` usage.

The following element defines the Liberty SSL key store:

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

Notes:

- The `<keyStore>` element is what provides a very basic self-signed SSL certificate that is just enough to allow the establishment of SSL between the client and the Liberty server.

Note: To make this work, you have to give your `batchManager` UNIX environment access to the `key.jks` file that is generated by Liberty. The easiest way to do this is to copy the `key.jks` file to some location accessible by the ID that runs `batchManager`, and exporting a JVM argument:

```
export JVM_ARGS="-Djavax.net.ssl.trustStore=/<path>/key.jks"
```

The WP102544 Techdoc "Quick Start Guide" provides detailed step-by-step instructions on locating that file, copying it, and setting the `JVM_ARGS` variable.

- In the next section ("batchManager access with SAF keyring for SSL " on page 26) we will show how to push that down into SAF.

The elements needed to enable SAF authentication and SAF authorization:

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

Notes:

- `<safRegistry>` enables SAF as the registry for authentication.
- `<safAuthorization>` enables SAF as the repository for role definitions.

Note: The element `racRouteLog="ASIS"` relates to IBM RACF. It enables the writing of ICH408I messages to the system console when an EJBROLE check does not succeed. Absent the `racRouteLog="ASIS"` element, role violations may occur but no ICH408I messages will be seen.

- `<safCredentials>` establishes the "profile prefix" to be used, in this example the value is `BBGZDFLT`. The profile prefix is related to the EJBROLE profiles that will be checked.

When you start your server, the `messages.log` file will give you key information about your environment:

```
CWWKB0103I: Authorized service group LOCALCOM is not available.
CWWKB0103I: Authorized service group SAFCRED is available.
CWWKB0103I: Authorized service group TXRRS is not available.
CWWKB0103I: Authorized service group WOLA is not available.
CWWKB0103I: Authorized service group ZOSDUMP is not available.
CWWKB0103I: Authorized service group ZOSWLM is not available.
CWWKB0104I: Authorized service group PRODMGR is not available.
CWWKB0104I: Authorized service group ZOSAIO is not available.
CWWKB0103I: Authorized service group CLIENT.WOLA is not available.
```

The "is available" messages indicate whether the server ID has READ to the SERVER profiles for the SAFCRED function¹².

Finally, this message indicates what features are installed:

```
CWWKF0012I: The server installed the following features: [ejbLite-3.2,
servlet-3.1, ssl-1.0, jndi-1.0, jca-1.7, batchManagement-1.0,
appSecurity-2.0, jdbc-4.1, batch-1.0, zosSecurity-1.0, json-1.0,
distributedMap-1.0].
```

EJBROLE profiles in support of batchManager usage

There are four application roles as part of this scenario; the following RACF commands would create all four of those roles:

```
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)
SETR RACLIST(EJBROLE) REFRESH
```

Notes:

- There authority provided by the three Java Batch roles are:

batchAdmin	Unrestricted access to all batch operations
batchSubmitter	Submit jobs, and perform operations on only jobs they submitted
batchMonitor	Read-only permission on all jobs

- EJBROLES are case-sensitive, so they must be defined as shown
- The "profile prefix" – `BBGZDFLT` in this example – may be whatever you wish. If something other than `BBGZDFLT`, then be sure to specify the profile prefix you used with:
`<safCredentials profilePrefix="<your_prefix_value>" />`

- If your APPL class is active you will also need an APPL profile created that matches your security prefix:

```
RDEF APPL BBGZDFLT OWNER(SYS1) UACC(READ)
SETR RACLIST(APPL) REFRESH
```

¹² If your server ID is granted READ to the other SERVER profiles, those functions will appear as "is available."

The final step is to grant READ to the IDs (or groups) you wish to have that application role:

```
PE BBGZDFLT.com.ibm.ws.management.security.resource.allAuthenticatedUsers
  CLASS (EJBROLE) ID (<id>) ACCESS (READ)
PE BBGZDFLT.com.ibm.ws.batch.batchAdmin CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchSubmitter CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchMonitor CLASS (EJBROLE) ID (<id>) ACCESS (READ)
SETR RACLIST (EJBROLE) REFRESH
```

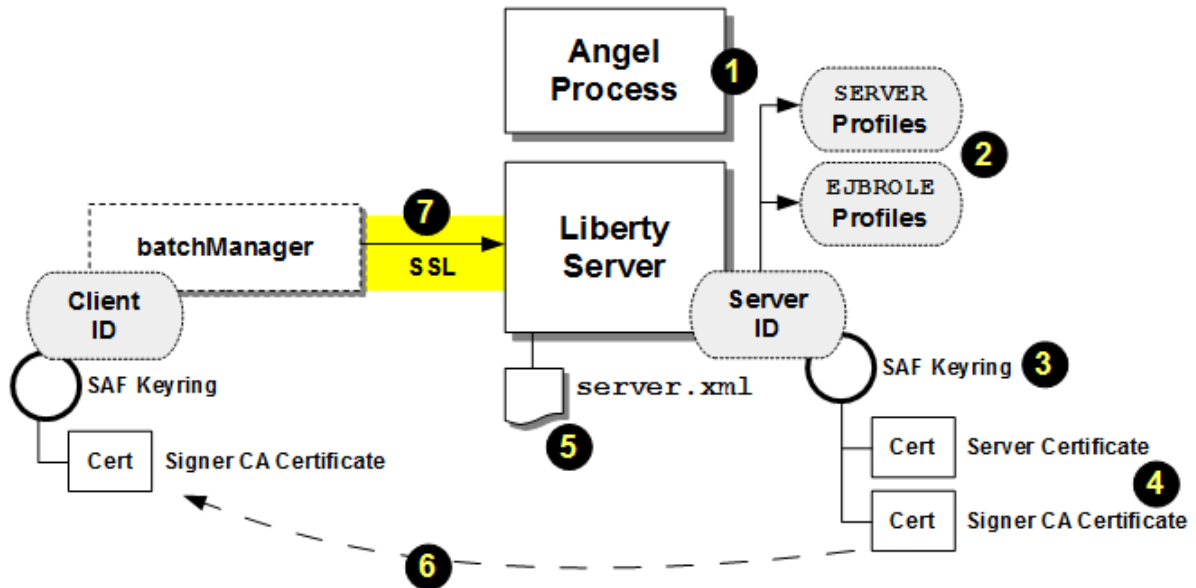
With that in place you can submit a job using batchManager and have SAF check authentication and provide EJBROLE authorization enforcement.

batchManager access with SAF keyring for SSL

In the previous section we used SAF for the user registry and for application role checking, but we left the SSL certificates in the default Liberty key file (key.jks). It's relatively simple to use that file and see things work, but it's not something you would use in a real-world, production environment.

In this section we will use SAF keyrings for the SSL key and trust store. All the other requirements are just like what was outlined in the previous section ("batchManager access with SAF registry and SAF role enforcement" starting on page 22).

The following picture provides a high-level overview of this setup:



Notes:

1. The Angel Process is not needed for SSL itself, but it is needed for EJBROLE checking. So it is present in this picture.
2. The SERVER and EJBROLE profiles for this scenario are exactly like what we saw in the previous scenario. We're changing the SSL keystore information, but the SERVER and EJBROLE checking is exactly the same.
3. Rather than a key file, we will use a SAF keyring. SAF keyrings are associated with a SAF ID, and they hold digital certificates.

- We are going to show the creation of a RACF Certificate Authority (CA), the creation of a server certificate, and the signing of the server certificate by the CA we create.

Note: This is still a "self-signed" certificate in the sense that the CA is *not* a "well known" CA. In a real-world environment you would have a well-known CA (such as Symantec) sign your server certificate. This will suffice for illustrating the RACF commands to create a server certificate, sign it, and connect it to a keyring.

- Updates to the `server.xml` file are made to tell Liberty about SAF as the key and trust store and the name of the keyring where the certificates are stored.
- The public key for CA certificate used to sign the server certificate is then made available to the client ID used for batchManager. That client can be on z/OS (what we're illustrating here), and therefore another keyring is involved. The CA public key certificate is connected to the client ID keyring. Or the client could be on a distributed server, in which case the CA public key certificate is exported from SAF, downloaded to the distributed server, and imported into the trustfile there. In either case the same thing is achieved: the client has the ability to trust the server's certificate because it has the CA certificate used to sign the server certificate.
- Now when the batchManager client initiates the connection to the Liberty, the server will pass its server certificate down to the client. The client can verify the trustworthiness of the certificate because it is signed by a CA it knows about (because the CA cert is in its keyring or trust file). The SSL session can be established.

We will illustrate this with the samples that follow¹³.

SERVER and EJBROLE profiles

These are exactly the same as illustrated under "batchManager access with SAF registry and SAF role enforcement" on page 22, and summarized under "Summary of updates: batchManager with SAF registry and SAF role enforcement" on page 60.

Creation of CA certificate, server certificate, and server keyring with certs

The following RACF command samples illustrate the creation of the certificates and keyring.

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty') OU('LIBERTY'))
  WITHLABEL('LibertyCA.LIBERTY') TRUST SIZE(2048) NOTAFTER(DATE(2018/12/31))
RACDCERT ID(<server_id>) GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com')
  O('IBM') OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY')
  SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY')) SIZE(2048)
  NOTAFTER(DATE(2018/12/30))
RACDCERT ID(<server_id>) ADDRING(Keyring.LIBERTY)
RACDCERT CONNECT(ID(<server_id>) LABEL('DefaultCert.LIBERTY')
  RING(Keyring.LIBERTY)) ID(<server_id>)
RACDCERT CONNECT(CERTAUTH LABEL('LibertyCA.LIBERTY')
  RING(Keyring.LIBERTY)) ID(<server_id>)
SETR RACLIST(DIGTCERT DIGTRING) REFRESH
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(<server_id>) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(<server_id>) ACCESS(READ)
SETR RACLIST(FACILITY) REFRESH
```

Notes:

- The `<server_id>` values would be the ID under which your Liberty server runs.
- The first RACDCERT command creates the CA certificate. This is *not* a well-known CA. If you had a certificate from a well-known CA you would import that into SAF and use that certificate to sign the server certificate (see the second RACDCERT command).
- The second RACDCERT command creates the server certificate and signs it with the CA certificate created with the first command. Again, if you had a well-known CA certificate

¹³ For a summary, see "batchManager access with SAF keyring for SSL " on page 26.

imported into SAF, you'd use that CA to sign the server certificate, not the CA with label 'LibertyCA.LIBERTY'.

The "Common Name" (CN) value is shown as the server host name for *our* test system. You would set that to *your* host name, and then match that host name value on the batchManager command.

- The third RACDCERT command creates a keyring and associates it with the server ID.
- The fourth and fifth RACDCERT commands connect the server certificate and CA certificates, respectively, to the server's keyring.
- The final set of commands provide the server ID the ability to list and access the keyrings, and then do a refresh on the FACILITY class.

Updates to server.xml in support of SAF keyring

The following line is **removed** from the `server.xml`:

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

The following feature is added:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
  <feature>ssl-1.0</feature>
</featureManager>
```

The following is also added:

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

Notes:

- The keyring name (`Keyring.LIBERTY` in this example) must match what was used on the RACDCERT command earlier to create the keyring. It is case sensitive.
- The password value is literally "password" as shown here¹⁴. Do not substitute any other value. Code it just as it appears here.

Client access to the CA certificate that signed the server certificate

The Liberty *server* is now set up with a keyring that contains a server certificate signed by the CA. When a *client* comes to this Liberty and attempts to create an SSL session, the server will present its server certificate. The client will need access to the CA certificate to know whether the server certificate can be trusted.

That means the client ID needs that CA certificate in either a keyring (z/OS SAF) or a trust file (z/OS or any other operating system).

¹⁴ Access to the keyring is protected by SAF based on the ID seeking access, not based on a password. The Java API requires a password to be supplied, even though it won't be used by SAF. So a hardcoded, literal value of "password" is used for Liberty.

File-based trust store

This involves exporting the CA certificate from SAF and importing it into the trust store file on the server where the batchManager client will be invoked.

The following is an example of the command to export the CA certificate from RACF:

```
RACDCERT CERTAUTH EXPORT (LABEL ('LibertyCA.LIBERTY'))
  DSN ('USER1.CERTAUTH.CRT') FORMAT (CERTDER)
```

Where the LABEL () value matches the label of the CA certificate in SAF, and the DSN () value is the z/OS dataset to which the certificate is exported.

The certificate can then be downloaded (in binary format) and imported into the trust file on the server. This is standard certificate management, using tools such as keytool or ikeyman. We're not going to show that here as our focus is z/OS.

z/OS keyring for ID

If you intend to operate batchManager on z/OS, then you can use a SAF keyring to hold the CA certificate.

Here we are going to illustrate the case where batchManager is run on z/OS with access to the SAF database where the CA certificate resides¹⁵. That could be on the same LPAR, or perhaps in the same Sysplex where the SAF database is shared across LPARs.

The following RACF sample illustrates how to create a keyring for the userid FRED and connect the CA certificate created earlier to the keyring:

```
RACDCERT ID (FRED) ADDRING (Keyring.FRED)
  RACDCERT CONNECT (CERTAUTH LABEL ('LibertyCA.LIBERTY')
  RING (Keyring.FRED)) ID (FRED)
SETR RACLIST (DIGTCERT DIGTRING) REFRESH
PERMIT IRR.DIGTCERT.LISTRING CLASS (FACILITY) ID (FRED) ACCESS (READ)
PERMIT IRR.DIGTCERT.LIST CLASS (FACILITY) ID (FRED) ACCESS (READ)
SETR RACLIST (FACILITY) REFRESH
```

The following UNIX environment variable sets JVM arguments that indicates where the keyring is. This is entered as one long command:

```
export JVM_ARGS="-Djavax.net.ssl.trustStore=safkeyring://FRED/Keyring.FRED
-Djavax.net.ssl.trustStoreType=JCERACFKS
-Djavax.net.ssl.keyStore=safkeyring://FRED/Keyring.FRED
-Djavax.net.ssl.keyStoreType=JCERACFKS
-Dcom.ibm.ssl.keyStoreFileBased=false
-Dcom.ibm.ssl.trustStoreFileBased=false
-Djava.protocol.handler.pkgs=com.ibm.crypto.provider
-Djavax.net.ssl.keyStorePassword=password"
```

Where:

- //FRED/Keyring.FRED is in this format: the keyring owner (//FRED) and the keyring name (/Keyring.FRED).
- keyStorePassword=password is literal; do not change the value "=password."

With that in place, you can open a UNIX shell under the ID of FRED, set the JVM arguments as shown, and enter a batchManager command. The server will send its server certificate, and FRED will have access to the CA cert in the Keyring.FRED keyring. The SSL session will be built.

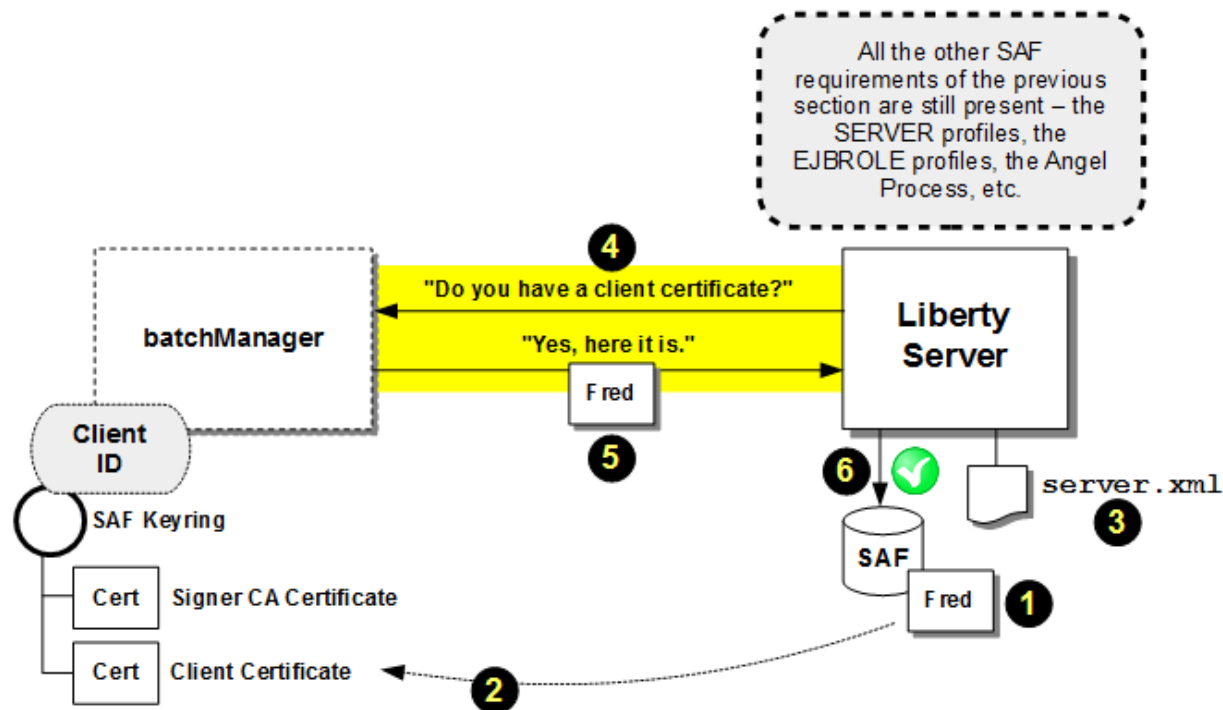
¹⁵ If on a different z/OS system, then it would be like a distributed server – export the CA certificate and move it to the z/OS system where the client will run. Then run the SAF command to import the certificate and connect it to a keyring.

batchManager access with SAF keyring for SSL and client certificate for authentication

Note: The information in this section assumes the previous section – "batchManager access with SAF keyring for SSL " starting on page 26 – is in place. This section provides an *addition* to the security configuration in support of SSL and SAF.

A client certificate eliminates the need to code the `--user` and `--password` on the batchManager command line. The client certificate is passed from the client to the Liberty server, and if the server trusts the certificate, then it trusts you are who you say you are.

The following picture illustrates the essentials of this:



Notes:

1. A client certificate is generated using a SAF command. The command indicates which SAF ID the certificate relates to. In the examples we'll show you, that SAF ID is `FRED`.
2. The client certificate is connected to the SAF keyring of the client ID.

Note: The certificate may also be exported from SAF and downloaded to a distributed server where it can be imported into a file-based keystore using a tool such as `keytool` or `keyman`.
3. Updates to the `server.xml` SSL section tell the Liberty server if client certificate authentication is supported, and if it is required.
4. When the client sends in its request to the server, the normal SSL establishment process occurs (the server sends its server certificate, and the client checks its CA signer certificates to see if it trusts the server certificate). Then, based on updates to the `server.xml` file (3), the server will request of the client its client certificate.
5. The client passes its client certificate up to the server over the SSL connection.
6. The server checks to see if the certificate against SAF to see if the ID represented in the certificate is a valid SAF ID, and if a matching certificate for that ID is found in SAF. If true, then the identity is trusted and the client is authenticated *without* a password.

Generate client certificate

The following RACF command illustrates how a client certificate can be generated for an ID. In this example the ID is `FRED`:

```
RACDCERT ID(FRED) GENCERT SUBJECTSDN(CN('Fred D. Client') O('IBM')
OU('LIBERTY')) WITHLABEL('FRED')
SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY')) SIZE(2048)
NOTAFTER(DATE(2018/12/30))
```

Note that Fred's client certificate was signed by a CA certificate – in this case it was the CA we created earlier, but it's a CA nevertheless. When this client certificate is received by the Liberty server, the certificate is trusted because it is signed by a CA known to SAF.

In reality client certificates would be signed a well-known CA certificate acquired by the organization and imported into SAF.

The following RACF command can be used to list certificates for Fred:

```
RACDCERT ID(FRED) LIST
```

The result would be:

```
Digital certificate information for user FRED:
```

```
Label: FRED
Certificate ID: 2QTG2cXExtnFxEBA
Status: TRUST
Start Date: 2016/10/08 00:00:00
End Date: 2018/12/30 23:59:59
Serial Number:
>02<
Issuer's Name:
>CN=CA for Liberty.OU=LIBERTY<
Subject's Name:
>CN=Fred D. Client.OU=LIBERTY.O=IBM<
Signing Algorithm: sha256RSA
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
*** No rings associated ***
```

Connect client certificate to client ID (or export for download to other servers)

If the client resides on z/OS and has access to the SAF database, then you can connect the client certificate to the client keyring. In the case of Fred:

```
RACDCERT CONNECT(ID(FRED) LABEL('FRED') RING(Keyring.FRED)) ID(FRED)
```

Then if you issue:

```
RACDCERT ID(FRED) LIST
```

You would see:

```
Digital certificate information for user FRED:
```

```
Label: FRED
Certificate ID: 2QTG2cXExtnFxEBA
Status: TRUST
Start Date: 2016/10/08 00:00:00
End Date: 2018/12/30 23:59:59
Serial Number:
>02<
Issuer's Name:
>CN=CA for Liberty.OU=LIBERTY<
Subject's Name:
>CN=Fred D. Client.OU=LIBERTY.O=IBM<
```

```

Signing Algorithm: sha256RSA
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
  Ring Owner: FRED
  Ring:
    >Keyring.FRED<

```

Or, if you prefer, you can export the certificate for download to another server:

```

RACDCERT ID(FRED) EXPORT(LABEL('FRED')) DSN('USER1.FRED.P12')
FORMAT(PKCS12DER) PASSWORD('secret')

```

That data set can be downloaded in binary format and the certificate can be imported into a key file on another server using a tool such as keytool or ikeyman. The password of the PKCS12 file is what's supplied in the `PASSWORD()` parameter, which is case sensitive.

Note: For another discussion of client certificate authentication with batchManager, see: <https://developer.ibm.com/wasdev/docs/using-ssl-certificate-authentication-batchmanager-liberty/>

Updates to `server.xml` to support client certificate authentication

This is the same as we saw in the previous section ("batchManager access with SAF keyring for SSL " starting on page 26) with two lines added:

```

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

```

Notes:

- The element `clientAuthenticationSupported="true"` tells Liberty that if the client asserts a client certificate, then Liberty is to accept it and check with SAF to see if it's trusted and can substitute for `userid` and `password` on the command line.
- The element `clientAuthentication="false"` tells Liberty that client certificates are not *required*. That is, if a client *without* a client certificate presents itself with `--user` and `--password`, then Liberty is to accept the credentials passed on the command line. That is useful for a server that will be accepting requests from both client certificates as well as credentials on the command.

Alternatively, if you set that value to `"true"`, then Liberty would *require* a client certificate to authenticate. The lack of a client certificate would be rejected, regardless of ID and password credentials passed on the command line.

Client-side JVM arguments and batchManager command example

This is almost the same as we saw for SAF-based SSL without client certificate authentication, with the difference here being Fred's client certificate is now in his keyring. But the same `JVM_ARGS` are set in the UNIX environment:

The following UNIX environment variable sets JVM arguments that indicates where the keyring is. This is entered as one long command:

```
export JVM_ARGS="-Djavax.net.ssl.trustStore=safkeyring://FRED/Keyring.FRED
-Djavax.net.ssl.trustStoreType=JCERACFKS
-Djavax.net.ssl.keyStore=safkeyring://FRED/Keyring.FRED
-Djavax.net.ssl.keyStoreType=JCERACFKS
-Dcom.ibm.ssl.keyStoreFileBased=false
-Dcom.ibm.ssl.trustStoreFileBased=false
-Djava.protocol.handler.pkgs=com.ibm.crypto.provider
-Djavax.net.ssl.keyStorePassword=password"
```

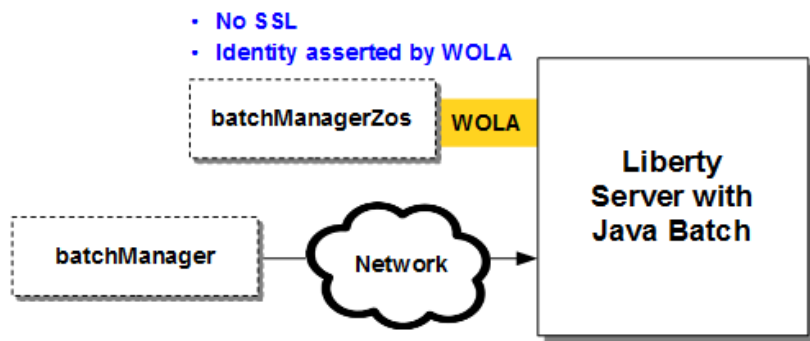
Where:

- //FRED/Keyring.FRED is in this format: the keyring owner (//FRED) and the keyring name (/Keyring.FRED).
- keyStorePassword=password is literal; do not change the value "=password."

The batchManager command can be issued *without* the --user and --password parameters. The server will ask the client for his client certificate; the client will go its keyring and send in the "FRED" client certificate; SAF will see it is signed by a CA it knows about; the client will be authenticated under that ID.

Differences between batchManager and batchManagerZos

The batchManagerZos command line client is *similar* to batchManager in that they both provide a command line interface for the submission and management of jobs. But they are *different* in the protocol used to communicate with the Liberty server, and that introduces some differences in the way security is configured:



- SSL considerations (certificates)
- Need to provide authentication credentials (ID/PW)
- If SAF, need APPL profile and awareness of 'unauthenticated guest'

We will work through the requirements of batchManagerZos in the sections that follow.

batchManagerZos access with basic registry and basic role enforcement

This can be made to work, but in general we encourage the use of SAF with batchManagerZos for authentication and authorization rather than "basic" definitions in the `server.xml` file.

The batchManagerZos command line client uses WOLA to communicate with the Liberty server. It will automatically assert the identity on the thread of the batchManagerZos client, which will be a SAF ID. However, based on the way WOLA works, that ID ends up being "null" to the Liberty basic authentication and basic authorization function, not the ID of "Fred" (or whatever) you expected to come over.

There are two ways you can make job submission work with batchManagerZos and without involving SAF:

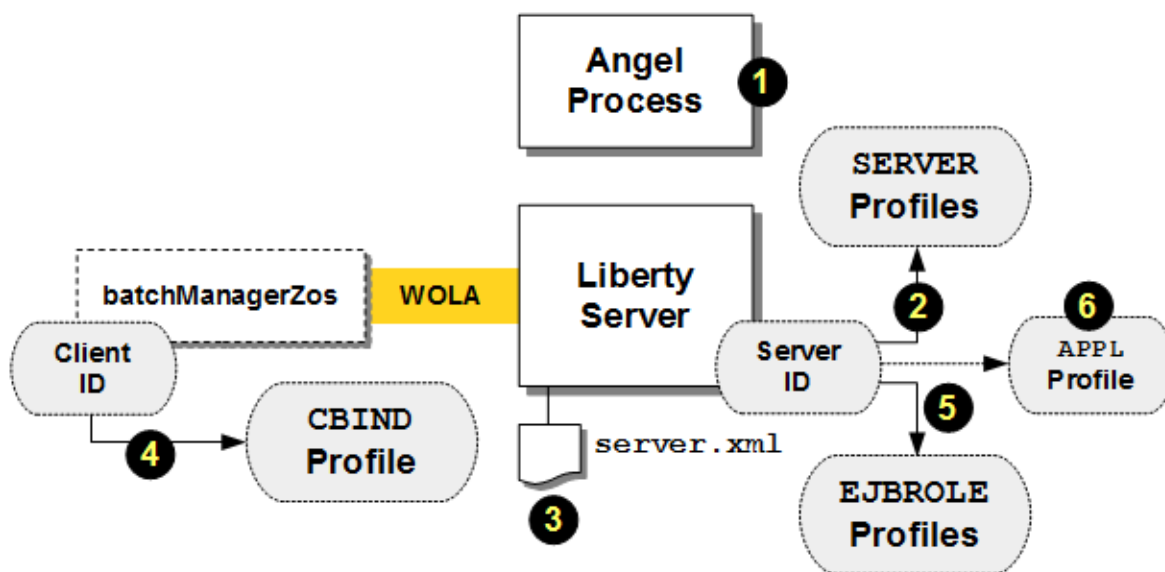
1. Remove the `appSecurity-2.0` feature, which turns off authentication and role checking, or
2. Code the following in the `server.xml` file:

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

The special-subject of `EVERYONE` permits any ID, including "null", to be allowed to submit a batch job. Acceptable for development and test, but not for anything requiring more stringent security.

batchManagerZos access with SAF registry and SAF role enforcement

The `batchManagerZos` command line client uses WOLA to communicate with the Liberty z/OS server, and WOLA brings a set of security requirements onto the table:



Notes:

1. To use WOLA, the Angel process must be started.
2. The ID (or group) assigned to the Liberty Server needs to have READ to a set of SAF `SERVER` profiles that grant access to the Angel and to the WOLA authorized services
3. The `server.xml` file needs to have a few definitions added enable WOLA and define the "three part name" used by the client when connecting to the server using WOLA.
4. The client ID needs READ access to a `CBIND` profile based on the "three part name" defined in the Liberty server. That's what grants the ID authority to "register into" the Liberty server.
5. The ID that is asserted over WOLA is checked against `EJBROLE` profiles to establish whether it has the authority to use the WebSphere Java Batch function.
6. If your `APPL` class is active, then an `APPL` profile matching the security prefix used on the `EJBROLE` profiles will be needed.

We'll illustrate each with the samples that follow¹⁶.

¹⁶ For a summary, see "Summary of updates: `batchManagerZos` and SAF" on page 58.

SERVER profiles in support of WOLA

Here's a list of the `SERVER` profiles related to Liberty z/OS. Not all of these are required to use WOLA (we will explain what's need in a bit); we are showing you the full list to give you a sense for what the `SERVER` profiles are used for.

BBG.ANGEL

BBG.AUTHMOD.BBGZSAFM

Controls access to the Angel process and the authorized services protected by the Angel process

BBG.AUTHMOD.BBGZSAFM.LOCALCOM

BBG.AUTHMOD.BBGZSAFM.WOLA

BBG.AUTHMOD.BBGZSCFM

BBG.AUTHMOD.BBGZSCFM.WOLA

Controls access to the WOLA function. Note the subtle difference in the third qualifier name.

BBG.AUTHMOD.BBGZSAFM.SAFCRED

BBG.SECPFY.BBGZDFLT

Controls access to SAF for certain security functions, most notably EJBROLE checking.

BBG.AUTHMOD.BBGZSAFM.TXRRS

BBG.AUTHMOD.BBGZSAFM.ZOSDUMP

BBG.AUTHMOD.BBGZSAFM.ZOSWLM

BBG.AUTHMOD.BBGZSAFM.ZOSAIO

Controls access to other functions – RRS, DUMP, WLM, and Asynch I/O.

There are two steps to this – (1) creating the `SERVER` profiles; and (2) granting `READ` to the ID (or group) you wish to have access to the authorized services.

To *create* the profiles, the following commands illustrate the syntax:

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.SECPFY.BBGZDFLT UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE) OWNER(SYS1)
```

Note: The `UACC(NONE)` means the "universal access" is "none" – that is, by default nobody can access. So the *existence* of a profile does *not* grant the access. Access is granted with a later `PERMIT` command that grants and ID or group the `READ` access.

You may wish to consider creating *all* the profiles even though you are only using *some* for WOLA. The presence of those yet unused profiles with `UACC(NONE)` would cause no harm.

To grant an ID (or group) the ability to access WOLA would imply the following commands:

```
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS(SERVER) ACCESS(READ) ID(<id>)
```



```
PERMIT BBG.AUTHMOD.BBGZSCFM CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS (SERVER) ACCESS (READ) ID (<id>)
```

In addition, two other SERVER profiles are need to support SAF authentication and authorization:

```
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.SECPFX.BBGZDFLT CLASS (SERVER) ACCESS (READ) ID (<id>)
```

The value `BBGZDFLT` is highlighted because that is a configurable value. The string `BBGZDFLT` is the default value for the "security prefix," though you may choose to use a different string. The value here relates to the prefix used on the `EJBRROLE` profiles, which are examined a bit later.

Finally, a refresh is needed:

```
SETROPTS RACLIST (SERVER) REFRESH
```

server.xml updates in support of WOLA

The following `server.xml` elements are needed to support use of `batchManagerZos` with SAF authentication and SAF role checking.

In the `<featureManager>` section of the file:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>zosLocalAdapters-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
</featureManager>
```

Notes:

- The `batch-1.0` and `batchManagement-1.0` elements are needed to implement the Java batch function.
- The `zosLocalAdapters-1.0` feature implements WOLA support.
- The `appSecurity-2.0` feature is what enables role checking.
- The `zosSecurity-1.0` feature is what enables SAF as the registry and role repository.
- Other features may be present; these are the essential features related to SAF authentication and authorization for `batchManagerZos` usage.

The elements needed to enable SAF authentication and SAF authorization:

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

Notes:

- `<safRegistry>` enables SAF as the registry for authentication.
- `<safAuthorization>` enables SAF as the repository for role definitions.

Note: The element `racRouteLog="ASIS"` relates to IBM RACF. It enables the writing of ICH408I messages to the system console when an `EJBRROLE` check does not succeed. Absent the `racRouteLog="ASIS"` element, role violations may occur but no ICH408I messages will be seen.

- `<safCredentials>` establishes the "profile prefix" to be used, in this example the value is `BBGZDFLT`. The profile prefix is related to the `EJBRROLE` profiles that will be checked.

Finally, the WOLA definition of the "three part name" that is used on the `batchManagerZos` command line to submit a job:

```
<zsoLocalAdapters wolaGroup="LIBERTY"
  wolaName2="BATCH"
  wolaName3="MANAGER"/>
```

Notes:

- The values are configuration; they do not need to be LIBERTY, BATCH, and MANAGER. They may be any uppercase, maximum 8 character values.
- The three-part name sequence must be unique on the LPAR.
- The value you code here is related to the CBIND profile that protects what IDs are allowed to execute a WOLA register operation into this server. That is explained next.

When you start your server, the `messages.log` file will give you key information about your environment:

```
CWWKB0103I: Authorized service group LOCALCOM is available.
CWWKB0103I: Authorized service group SAFCREd is available.
CWWKB0103I: Authorized service group TXRRS is not available.
CWWKB0103I: Authorized service group WOLA is available.
CWWKB0103I: Authorized service group ZOSDUMP is not available.
CWWKB0103I: Authorized service group ZOSWLM is not available.
CWWKB0104I: Authorized service group PRODMGR is not available.
CWWKB0104I: Authorized service group ZOSAI0 is not available.
CWWKB0103I: Authorized service group CLIENT.WOLA is available.
```

The "is available" messages indicate whether the server ID has READ to the SERVER profiles for those functions. The messages highlighted in yellow are for SAF access and WOLA¹⁷.

The following message indicates what "three part name" the server is using:

```
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered
with the Liberty profile server using the following name: LIBERTY BATCH
MANAGER
```

Finally, this message indicates what features are installed:

```
CWWKF0012I: The server installed the following features: [ejbLite-3.2,
servlet-3.1, ssl-1.0, jndi-1.0, jca-1.7, batchManagement-1.0,
appSecurity-2.0, jdbc-4.1, zosLocalAdapters-1.0, batch-1.0,
zosSecurity-1.0, json-1.0, distributedMap-1.0].
```

CBIND profile in support of WOLA

The CBIND class is used to protect what IDs may perform a WOLA "registration" into the Liberty server. The CBIND profile is based on the three-part name used by the Liberty server. The following commands create the CBIND profile and permit a user access:

```
RDEFINE CBIND BBG.WOLA.LIBERTY.BATCH.MANAGER UACC(NONE) OWNER(SYS1)
PERMIT BBG.WOLA.LIBERTY.BATCH.MANAGER CLASS(CBIND) ACCESS(READ) ID(<id>)
SETROPTS RACLIST(CBIND) REFRESH
```

Notes:

- The highlighted values in the first line represent the "three part name" defined in the Liberty server's `server.xml` file.
- The ID specified on the second line is the ID under which the `batchManagerZos` command line client runs. If you submit that command from a UNIX shell, then the ID is the ID you used to log onto the shell. If you submit that command using JCL and `BPXBATCH`, then it's the ID under which the JCL job executes.

Note: The ID (<id>) value may specify a SAF GROUP name if you wish. Then any ID that is a member of that group would be given access.

¹⁷ If you granted your server ID READ to all the SERVER profiles, you would see all those as "is available."

- The CBIND profile can be wildcarded to allow multiple Liberty serves with slightly different three-part names to be controlled under a single CBIND. Check the SAF documentation for wildcard rules and test to insure expected operations.

EJBROLE profiles in support of batchManagerZos usage

There are three application roles supported by the product:

batchAdmin	Unrestricted access to all batch operations
batchSubmitter	Submit jobs, and perform operations on only jobs they submitted
batchMonitor	Read-only permission on all jobs

The following RACF commands would create all three of those roles:

```
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)
SETR RACLIST(EJBROLE) REFRESH
```

Notes:

- EJBROLES are case-sensitive, so they must be defined as shown
- The "profile prefix" – BBGZDFLT in this example – may be whatever you wish. If something other than BBGZDFLT, then be sure to specify the profile prefix you used with:

```
<safCredentials profilePrefix="<your_prefix_value>" />
```

- If your APPL class is active you will also need an APPL profile created that matches your security prefix:

```
RDEF APPL BBGZDFLT OWNER(SYS1) UACC(READ)
SETR RACLIST(APPL) REFRESH
```

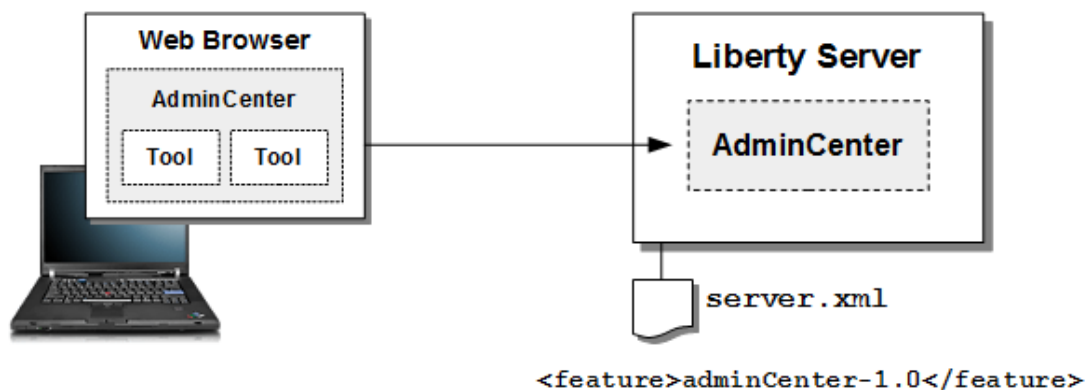
The final step is to grant READ to the IDs (or groups) you wish to have that application role:

```
PE BBGZDFLT.com.ibm.ws.batch.batchAdmin CLASS(EJBROLE) ID(<id>) ACCESS(READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchSubmitter CLASS(EJBROLE) ID(<id>) ACCESS(READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchMonitor CLASS(EJBROLE) ID(<id>) ACCESS(READ)
SETR RACLIST(EJBROLE) REFRESH
```

With that in place you can submit a job using batchManagerZos and have SAF check authentication and provide EJBROLE authorization enforcement.

AdminCenter overview

The AdminCenter provides a graphical management interface to Liberty, and by extension¹⁸, the Java Batch tool function that runs there:



¹⁸ Starting with Liberty 16.0.0.4.

Note: The Java Batch tool is capable of purging jobs (starting with 17.0.0.2), but it can't be used to submit jobs.

The security considerations are very similar to batchManager¹⁹:

- Encryption – an SSL session
- Authentication – checking whether the identity is valid
- Authorization – checking if the identity has authority to perform that function

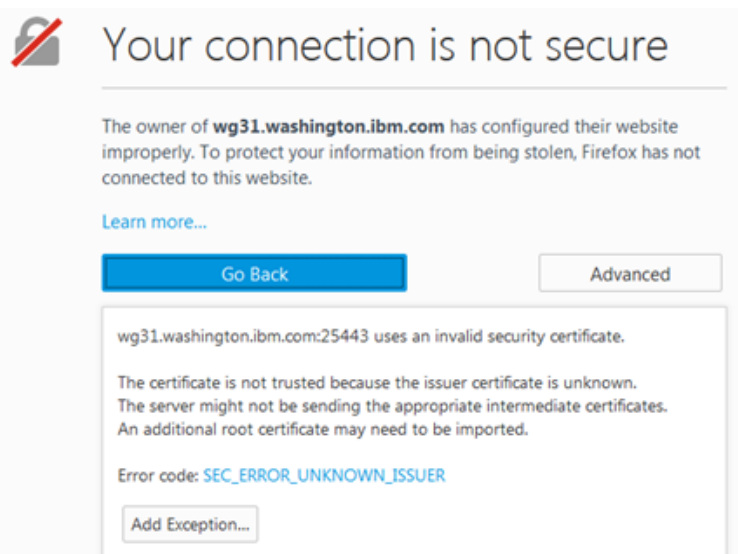
Some quick notes about the AdminCenter:

- It is enabled with the `adminCenter-1.0` feature in the `<featureManager>` list.
- Once enabled, it is access with the following URL:

`https://<host>:<port>/adminCenter/`

Where:

- The protocol `https` is used (to indicate SSL)²⁰
- The `<port>` value is the `https` port of the Liberty server
- If the server certificate is signed by a CA that is not known to the browser, it will challenge you:



We mention this because the basic Liberty `<keyStore>` certificate will be unrecognized by the browser, as will a certificate that is signed by a RACF CA and not a well-known CA.

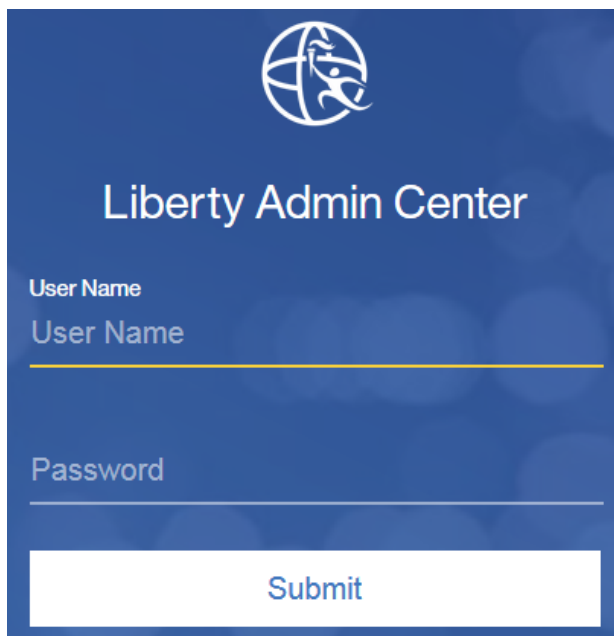
You can manually accept the challenge, or you can get the CA certificate from Liberty and import it into your browser security store. Each browser's steps to accomplish that are different, so we won't cover that in this document.

Note: If your Liberty server certificate is signed by a well-known CA, then you won't have this issue as your browser will recognize the CA and trust the server certificate.

¹⁹ Both are accessing via TCP and HTTP, so it makes sense they would have similar security considerations.

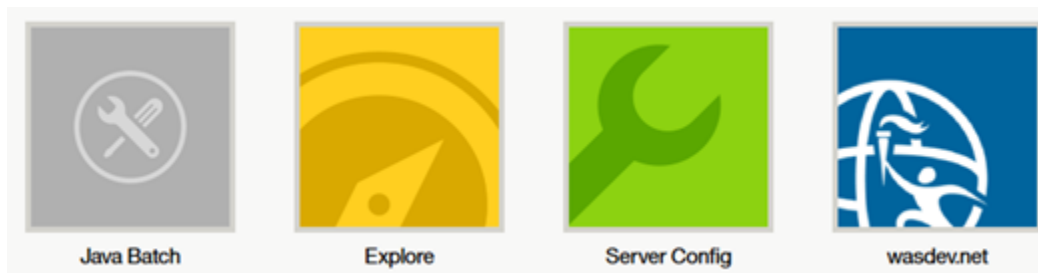
²⁰ You may use `http://` and point your browser at the non-SSL port. It will re-direct to `https://` and the SSL port.

- The login panel looks like this:



The "User Name" and "Password" must be found in either the basic registry in the `server.xml`, or in SAF, depending on which option you are using.

- If you are properly authenticated and the ID you use has been granted access to the Administrator role (more on that detail coming up), you will see:



The "Java Batch" tool is what is used to monitor the batch jobs.

AdminCenter access with basic registry and basic role checking

The simplest thing to do is what was done back under "batchManager access with basic registry and basic role enforcement" starting on page 19. Here we will illustrate a few additions to enable the AdminCenter and grant access.

The following `server.xml` example shows the basic security needed for *both* the AdminCenter and batchManager, with user "Fred" having access to the AdminCenter, with Mary and Bob granted access to batchManager:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>adminCenter-1.0</feature>
</featureManager>

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

<basicRegistry id="basic1" realm="jbatch">
```

```

<user name="Fred" password="fredpwd" />
<user name="Mary" password="marypwd" />
<user name="Bob" password="bobpwd" />
</basicRegistry>

<administrator-role>
  <user>Fred</user>
</administrator-role>

<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <user name="Mary" />
  </security-role>
  <security-role name="batchSubmitter">
    <user name="Bob" />
  </security-role>
</authorization-roles>

```

Notes:

- The `adminCenter-1.0` feature is what enables the AdminCenter in the Liberty server instance.
- The `<basicRegistry>` section defines the users. This can be for both users of the AdminCenter and batchManager. Fred, Mary and Bob are defined here.
- The `<administrator-role>` section defines which users have administrator access to the AdminCenter. In this example, only Fred has administrator access.
- The `<authorization-roles>` section defines which users have access to Java Batch functions. In this example, Mary has Admin and Bob has Submitter.

With that in place, you can access the AdminCenter.

AdminCenter access with SAF registry and SAF role enforcement

This is similar to what we illustrated under "batchManager access with SAF registry and SAF role enforcement" starting on page 22. Here we will illustrate one `server.xml` addition to enable the AdminCenter, and one additional EJBROLE definition to define and grant access to the Liberty administrator role.

The `server.xml` elements to support both AdminCenter and batchManager using SAF registry and SAF role enforcement would be:

```

<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
  <feature>adminCenter-1.0</feature>
</featureManager>

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

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

```

Note: Here we are still using the "basic" Liberty keystore and self-signed certificate for SSL.

In addition to the EJBROLES for batchManager access defined in the earlier section (starting on page 22), one additional EJBROLE is needed to define administrator access to the AdminCenter:


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

Then you grant READ to whatever IDs (or groups) you wish to have administrator access:

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

With that in place, you can access the AdminCenter.

AdminCenter access with SAF keyring for SAF SSL

This is similar to what we illustrated under "batchManager access with SAF keyring for SSL " starting on page 26. Back in that section we explained how to update the `server.xml` to support SAF-based SSL, and how to create a server certificate and sign it with a CA certificate.

The XML updates to support SAF-based SSL for browser access to the AdminCenter are:

The following line is **removed** from the `server.xml`:

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

The following features are **added**:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
  <feature>ssl-1.0</feature>
  <feature>adminCenter-1.0</feature>
</featureManager>
```

The following section is added to defined SAF as the key and trust store for SSL:

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

In addition to the EJBROLES for batchManager access defined in the earlier section (starting on page 26), one additional EJBROLE is needed to define administrator access to the AdminCenter:

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

Then you grant READ to whatever IDs (or groups) you wish to have administrator access:

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

With that in place, you can access the AdminCenter.

Securing the UNIX file system

This is a topic that can be relatively simple, or more sophisticated, depending on a number of factors related to the file ownership and started task ID you use for Liberty z/OS.

The following Techdoc has more information on this:


<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102687>

Specifically, the "Security Overview" unit that can be found there.

Security Overview

Security is always an important topic, and that is true with Liberty z/OS as well.

In this unit we focus on three layers of security: file system security; z/OS server security; and Java application-layer security. We cover recommended good practices, and provide an understanding how z/OS Security Access Facility (SAF, the security interface provided with z/OS, behind which a security product such as IBM RACF runs) is used with Liberty z/OS.



[Liberty zOS - Security Overview - CHARTS.pdf](#)



[Liberty zOS - Security Overview - NOTES.pdf](#)

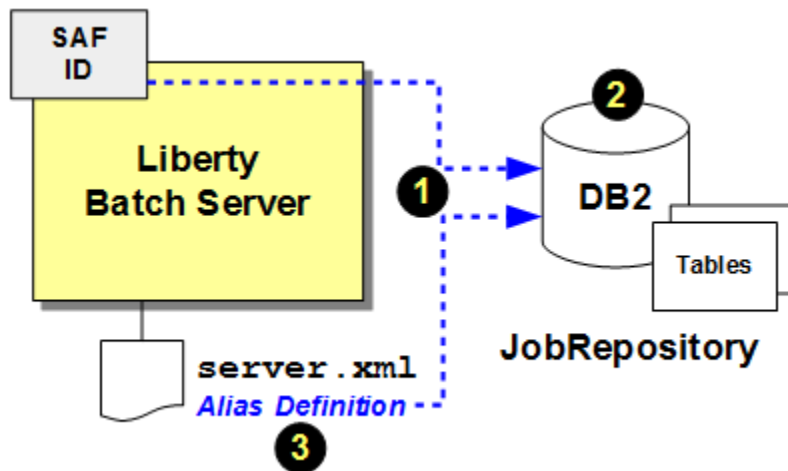
The section titled "File Related Security" covers this in some depth of detail.

Securing the JobRepository database (DB2)

Underlying the Liberty Batch function is a relational database that holds information about the batch jobs that have been run, as well as information about jobs that are currently running. This relational database can take several forms, depending on which you choose:

In-memory	The easiest, but also the least robust. This is only for development and light-duty testing.
File-based	For example, a file-based database product like Derby. This is also intended for development or test, but not for production on z/OS.
Relational DB product	For example, DB2 z/OS. This is our focus for this section of the document.

The security around the JobRepository DB2 can be broken down into three parts:



Notes:

1. The ID that tries to access the DB2 JobRepository tables. This depends on whether the JDBC connection is Type 2 (cross-memory) or Type 4 (TCP-based).
2. The access privileges granted to the database tables that make up the JobRepository
3. If JDBC Type 4, then the way the authentication alias password is protected in the configuration file

Identity asserted into DB2

This depends on the type of connection you have configured:

JDBC Type 2	The identity of the server ²¹ is asserted over the cross-memory connection to DB2.
JDBC Type 4	The identity specified on an authentication alias definition is asserted over the TCP/IP connection to DB2.

For JDBC Type 2, the "identity of the server" is the started task ID assigned to the Liberty server that runs Liberty Batch. This is covered in some detail under "SAF in support of Liberty server started task" starting on page 18.

For JDBC Type 4, the "identity specified on an authentication alias definition" is what's configured in the `server.xml`, for example:

```
<authData id="batchAlias" user="BATCHID" password="BATCHPW" />

<dataSource id="batchDB"
  containerAuthDataRef="batchAlias"
  type="javax.sql.XADataSource"
  jdbcDriverRef="DB2T4">
  <properties.db2.jcc
    serverName="myserver.com"
    portNumber="9446"
    databaseName="DB2LOC"
    driverType="4" />
</dataSource>
```

In both cases – Type 2 or Type 4 – the ID that comes across into DB2 will need to have the appropriate access privileges defined or the batch job execution will not succeed.

Access privileges on the DB2 tables

DB2 for z/OS protects what IDs may access database tables, and what privileges they have if they are allowed to access. Absence of the proper authority results in error messages in the Liberty `messages.log` file, such as:

```
Internal Exception: com.ibm.db2.jcc.am.SqlSyntaxErrorException: <ID> DOES
NOT HAVE THE PRIVILEGE TO PERFORM OPERATION SELECT ON OBJECT
JBATCH.JOBEXECUTION. SQLCODE=-551, SQLSTATE=42501, DRIVER=4.14.119
```

Where `<ID>` is the identity asserted into DB2, either the Liberty server started task ID (if JDBC Type 2), or the authentication alias (if JDBC Type 4).

The authority the ID needs is `SELECT`, `INSERT`, `DELETE` and `UPDATE` on the four tables that make up the Liberty Batch JobRepository when DB2 z/OS is used. Granting this authority is done with `GRANT` commands:

```
GRANT SELECT, INSERT, DELETE, UPDATE ON
  TABLE <schema>.STEPTHREADINSTANCE TO <id>;
GRANT SELECT, INSERT, DELETE, UPDATE ON
  TABLE <schema>.JOBEXECUTION TO <id>;
GRANT SELECT, INSERT, DELETE, UPDATE ON
```

²¹ Liberty also supports the assertion of the identity on the thread of execution, but that is an option we will set aside for this discussion of batch processing.

```
TABLE <schema>.STEPTHREADEXECUTION TO <id>;
GRANT SELECT, INSERT, DELETE, UPDATE ON
TABLE <schema>.JOBINSTANCE TO <id>;
```

Where:

- `<schema>` is the schema value used when the tables were created. For example, JBATCH if you used the examples in the WP102544 Step-by-Step guide.
- `<id>` is the ID being asserted into DB2.

It is possible to grant the authority to a SAF *group*, and connect IDs to the group.

It is possible to make that one long command if you wish:

```
GRANT SELECT, INSERT, DELETE, UPDATE ON
TABLE <schema>.STEPTHREADINSTANCE,
      <schema>.JOBEXECUTION,
      <schema>.STEPTHREADEXECUTION,
      <schema>.JOBINSTANCE TO <id>;
```

Authentication alias password encoding

If JDBC Type 4 is the connection type you're using to get into DB2, then the ID and password will be included as an "authentication alias" definition in the Liberty `server.xml` configuration, for example:

```
<authData id="batchAlias" user="BATCHID" password="BATCHPW" />

<dataSource id="batchDB"
  containerAuthDataRef="batchAlias"
  type="javax.sql.XADataSource"
  jdbcDriverRef="DB2T4">
  <properties.db2.jcc
    serverName="myserver.com"
    portNumber="9446"
    databaseName="DB2LOC"
    driverType="4" />
</dataSource>
```

The example here shows the password "in the clear," which means anyone with access to the configuration file can see it. If you have your file permissions²² properly secured, that may prove sufficient.

But you may wish to encrypt (or "encode") the password string so its true value is hidden from view. This can be done with the `securityUtility` function provided by Liberty itself. This utility can be found in the `/bin` directory where Liberty is installed.

Three forms of password string encoding are supported:

xor	A relatively simple encoding scheme. It will keep casual observers from knowing the true password, but it will not keep those seeking to know the password ²³ from finding out.
aes	An encryption algorithm which requires a key to encrypt <i>and</i> decrypt. This means you must provide the Liberty server with knowledge of the key.
hash	An encryption algorithm that produces a lengthy string without need of a key.

Encoding with xor

The following is an example of using the `securityUtility` function to `xor` encode a password string:

²² See "Securing the UNIX file system" on page 43.

²³ There are websites that can decipher an xor string.

```
> export JAVA_HOME=/usr/lpp/java/J8.0_64
> ./securityUtility encode --encoding=xor BATCHPW
{xor}HR4LHBcPCA==
>
```

You then copy that string and supply it as the password on the authentication alias:

```
<authData id="batchAlias" user="BATCHID" password="{xor}HR4LHBcPCA==" />

<dataSource id="batchDB"
  containerAuthDataRef="batchAlias"
  type="javax.sql.XADataSource"
  jdbcDriverRef="DB2T4">
  <properties.db2.jcc
    serverName="myserver.com"
    portNumber="9446"
    databaseName="DB2LOC"
    driverType="4" />
</dataSource>
```

Encoding with aes

The process is very similar to `xor`, except you supply a key value to the encoding, and you have to let Liberty know what that key is:

```
> export JAVA_HOME=/usr/lpp/java/J8.0_64
> ./securityUtility encode --encoding=aes --key=secretkey batchidpw
{aes}AE0kRVaS+xxV0z13VEv5ngMhakK31tke5GGOctjZxniK
>
```

You then copy that string and supply it as the password on the authentication alias:

```
<authData id="batchAlias" user="BATCHID"
  password="{aes}AE0kRVaS+xxV0z13VEv5ngMhakK31tke5GGOctjZxniK" />

<dataSource id="batchDB"
  containerAuthDataRef="batchAlias"
  type="javax.sql.XADataSource"
  jdbcDriverRef="DB2T4">
  <properties.db2.jcc
    serverName="myserver.com"
    portNumber="9446"
    databaseName="DB2LOC"
    driverType="4" />
</dataSource>
```

And you must provide the Liberty server knowledge of that key. There are two ways to accomplish this:

- With a variable definition in the XML:

```
<variable name="wlp.password.encryption.key" value="secretkey" />
```

Provide that in the XML at some point above the encrypted `password=` string.

If you wish, you can create a separate file for that key and use `<include>` processing to bring that value into the `server.xml`. For example, let's say you coded that `<variable>` tag in a separate file called `key.xml`:

```
<server>
  <variable name="wlp.password.encryption.key"
    value="secretkey" />
</server>
```

Then in your `server.xml` you would provide:

```
<include location="/<path>/key.xml" />
```

That would bring in the externalized XML with the key. You would consider this approach if your `server.xml` has broader "read" permissions than you'd like for encryption keys like this. You would lock down the permissions on the `key.xml` file to READ for *just* the server ID and nobody else.

- With a variable defined in the `bootstrap.properties` file

```
wlp.password.encryption.key = secretkey
```

This file can be secured with UNIX file permissions so only the server ID has READ to it.

Encoding with hash

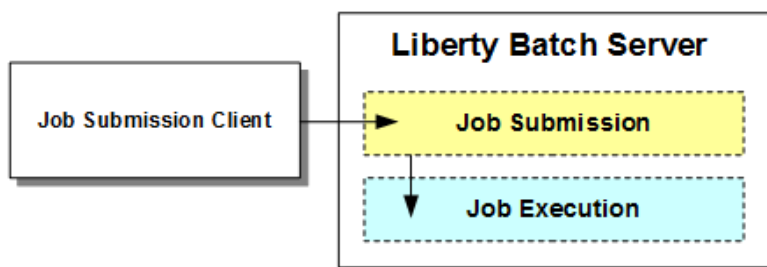
This method of password encoding is not supported for JDBC Type 4 authentication alias passwords²⁴.

²⁴ It works for encoding passwords for userids in a "basic registry," but not for JDBC Type 4 passwords.

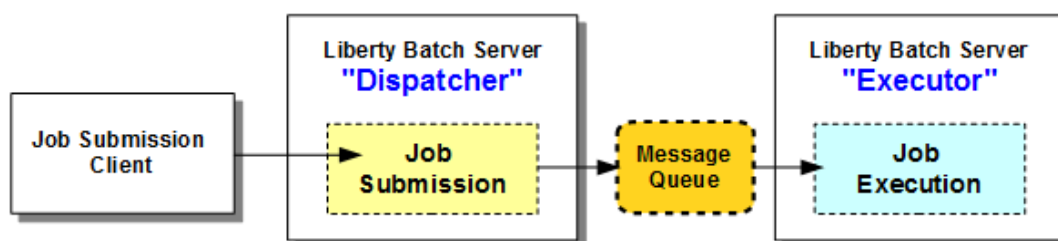
Multi-Server Environment

Overview – setting in context to the single server information

Up to this point in the document we have assumed a single server topology; that is, the job submission and job execution takes place in the same server:



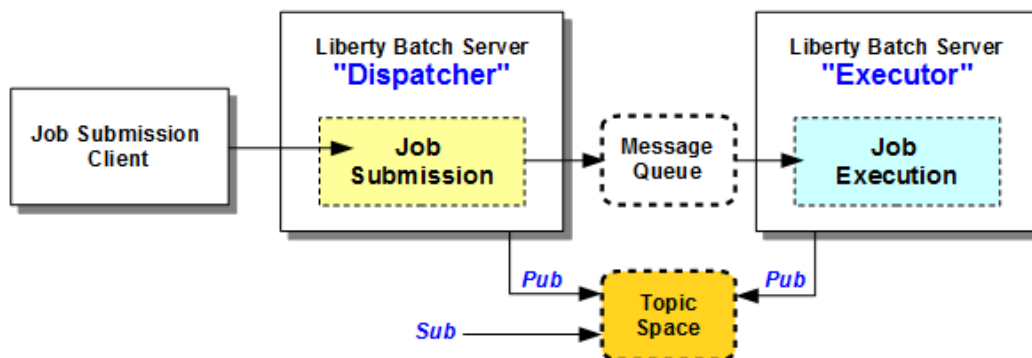
The alternative topology approach uses multiple Liberty servers, and separates the job submission task from the job execution task²⁵:



When you have a "Dispatcher" and "Executor" topology like this, you *must* have a message queue between the two, either MQ or the built-in messaging of Liberty, sometimes known as the "default messaging provider"²⁶. A single server topology requires no queue.

From a security perspective, our focus is on the "Message Queue" part of that picture. That queue can be secured.

There is another "message queueing" element to this, and it is separate from the queue used to pass job execution requests. This other element is a "pub/sub topic space," which has processes that *publish* to the topic, and processes that *subscribe* to the topic:



A few notes about this:

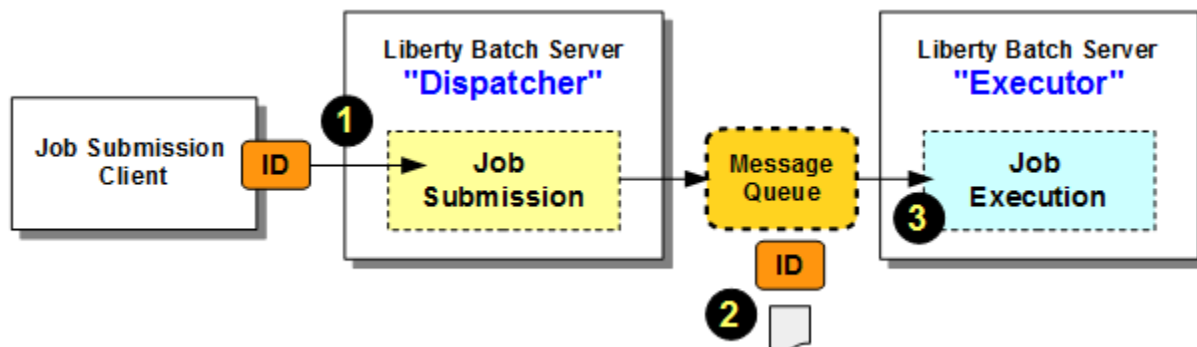
- It's optional; you don't have to make use of this pub/sub capability
- The pub/sub mechanism can be used with a single server topology or a multi-server topology.

²⁵ There are variations on this basic picture. See the WP102626 document at ibm.com/support/techdocs for more.

²⁶ Some may still refer to it as the "Service Integration Bus," or "SIBus" for short. That's no longer the "official" name.

Job submission identity when multi-server is used

When a multi-server environment exists, authentication and authorization takes place in the Dispatcher server. If the identity seeking to submit a job can't be authenticated, or is authenticated but does not have authority, then the request is rejected. If the identity is accepted and the job is submitted, then the message goes on the queue along with the identity the Dispatcher authenticated:

**Notes:**

1. The job submission client passes its identity to the Dispatcher, and the Dispatcher authenticates and authorizes. This mechanism is the same as was covered under the "Single-Server Environment" section starting on page 18.
2. Assuming the user was authenticated and authorized, the Dispatcher places the job submission message on the queue, along with the authenticated ID name value.
3. The Executor that picks up the message reads the ID value *and authenticates / authorizes the user again.*

The italicized part of Note #3 is the key. It means the Executor server must be configured to handle authentication and authorization in a way compatible with the Dispatcher server.

General Rule: If you use SAF authentication and authorization in your Dispatcher, do the same in your Executors. Or, if you use "basic" in your Dispatcher, use "basic" in your Executors. It is technically possible to mix the security mechanisms. But it introduces more things to think about.

Considerations when security is "basic"

Earlier we saw the "basic" security setup in the *Dispatcher* to be something like this:

```

<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
</featureManager>

<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>
  
```

That would allow "Fred" to authenticate and be authorized to the "batchAdmin" role.

For this to work in a multi-server model, the same `<basicRegistry>` and `<authorization-roles>` elements would need to be in the Executors as well. When the ID of "Fred" is passed over MQ to the Executor, it would need to check if "Fred" is a valid ID and had access to the proper role.

Note: Some of these "common configuration" elements *could* be stored in a central location and merged into each server's `server.xml` using the Liberty `<include>` function. That would create a single, central location for this "basic" security configuration. After the `<include>` merging is complete, the result is as we described above: each server would have a copy of it. The key point remains: the Executor needs to authenticate and authorize, and when "basic" security is used, the Executor needs access to the "basic" security definitions.

When `batchManagerZos` is used with "basic" security enabled, we have an additional thing to contend with: the identity using `batchManagerZos` (and WOLA) can't be determined by the basic security mechanism. To get around this we illustrated the use of the `<special-subject>` element in the `<security-role>`:

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

What that does is allow *anyone*, including a *null* ID (which is what basic security sees when `batchManagerZos` is used with basic security) to access.

Three points:

1. If you're using SAF for authentication and authorization, the ID under which you invoke `batchManagerZos` will be used. So it's not that `batchManagerZos` has no ability to understand the submitter ID, it's just the case when basic security is used.
2. If you're using `<special-subject type="EVERYONE"/>` in the Dispatcher basic security, have the same in the Executor `server.xml` as well.
3. When using `batchManagerZos` you probably should *not* use basic security except for development or simple testing. That `<special-subject>` element in the `batchAdmin` security role grants authority to *any* authenticated user.

Considerations when security is SAF

When the Dispatcher and all the Executors are part of the same Sysplex, sharing the same SAF database, then using SAF for authentication and authorization is relatively easy. The key consideration is making sure the `server.xml` definitions for all the servers specify what's needed for SAF, as was illustrated earlier in this document for the single server definition.

If the Dispatcher and Executors are on different systems using different (non-replicated) SAF databases²⁷, then the key would be to make sure the information in all SAF locations is the same.

Batch group authorization - 18.0.0.1

This function was made available in the 18.0.0.1 release.

The reason this function was developed was because the previous security model lacked sufficient granularity. An ID with `batchAdmin` could view or control batch jobs submitted by *any* ID, and an ID with `batchMonitor` could view batch jobs submitted by *any* ID. There was

²⁷ That is possible if you're using MQ and client mode connections between disparate z/OS systems.

a need to restrict this *admin* or *monitor* authority to a smaller set of jobs. This is done with the batch group authorization function.

Important: This new function applies only to a multi-server environment. The assignment of the group name to the submitted job occurs when a batch Executor server picks a job submission message off the message queue.

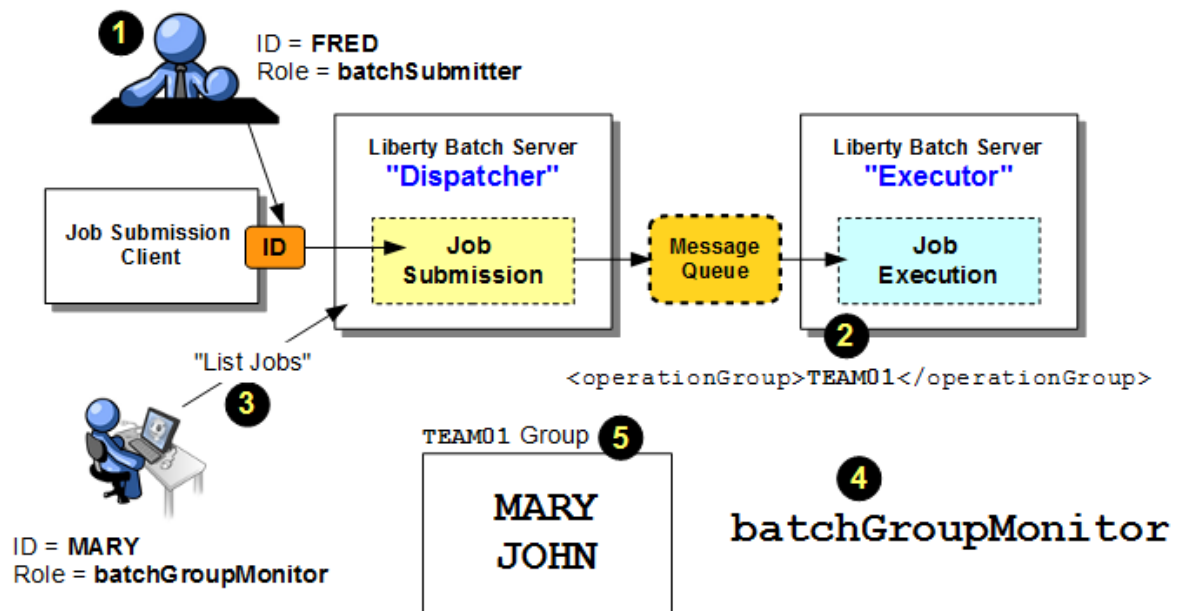
This new function enables the assignment of a group name to a job that is submitted and dispatched to an Executor server. The group name is then "associated" with the job instance, and both job instance and group name are maintained in a new JobRepository table called `GROUPASSOCIATION`.

Two new authorization roles are available:

<code>batchGroupAdmin</code>	Grants admin authority (read/write) for an ID associated with with this role.
<code>batchGroupMonitor</code>	Provides monitor authority (read) for an ID associated with this role.

Here's the key: when an ID is attempting to access a batch job that's been associated with a group name, the batch runtime will check to see if the ID is a member of the group *and* the ID has access to one of the two new authorization roles.

In picture form, it looks like this:



Notes:

1. A job is submitted to the Dispatcher. The ID of the submitter is `FRED`. Fred's ID has access to the `batchSubmitter` role that's been part of the product since the beginning.
2. In the Executor server a new sub-element is defined to the `<batchJmsExecutor>` definition. This new element defines an "operationGroup" name to be associated with this job. In this example we're using the name `TEAM01`. For example:

```
<batchJmsExecutor
  activationSpecRef="batchActivationSpecA"
  queueRef="batchJobSubmissionQueue">
  <operationGroup>TEAM01</operationGroup>
</batchJmsExecutor>
```

3. Another user -- `MARY` -- wants to check on the status of the jobs, including the job `FRED` submitted. `MARY` has access to the new `batchGroupMonitor` role, and `MARY` is also a member of the `TEAM01` group.

4. The batch runtime sees MARY's request to list jobs. The runtime knows about TEAM01 specified with <operationsGroup> because it can see that in the new GROUPASSOCIATION table. The batchGroupMonitor role²⁸ is checked to see if MARY is authorized to that role. She is.
5. The user registry is checked to see if the TEAM01 group exists, and if MARY is a member of that group. She is. MARY is allowed access to list FRED's job.

If the ID attempting access fails either of those checks -- access to the authorization role or a member of the associated group -- then access to the job is denied.

Updated JobRepository tables required

For this new function to work, a new JobRepository table is needed. The table name is GROUPASSOCIATION, and it may be created automatically (if you allow JPA to do so), or manually.

If you'd like to see the DDL to create the new table, you can run the ddlGen shell script against an 18.0.0.1 server. That will generate the DDL file, including the new table. You may inspect the new table definition and create using whatever mechanism you normally use to create tables in your relational database system.

Syntax in the server.xml <batchJmsExecutor> element

The <batchJmsExecutor> element typically looks something like this:

```
<batchJmsExecutor
  activationSpecRef="batchActivationSpecA"
  queueRef="batchJobSubmissionQueue">
</batchJmsExecutor>
```

Where activationSpecRef= and queueRef= point to other elements in the server.xml where the activation spec and the JMS queue are defined.

This new new function adds a sub-element to the <batchJmsExecutor> element. For example, to define a group called TEAM01, the XML would be updated with this:

```
<batchJmsExecutor
  activationSpecRef="batchActivationSpecA"
  queueRef="batchJobSubmissionQueue">
  <operationGroup>TEAM01</operationGroup>
</batchJmsExecutor>
```

It's possible to list multiple groups. An example of that would be:

```
<batchJmsExecutor
  activationSpecRef="batchActivationSpecA"
  queueRef="batchJobSubmissionQueue">
  <operationGroup>TEAM01</operationGroup>
  <operationGroup>TEAM02</operationGroup>
  <operationGroup>TEAM03</operationGroup>
</batchJmsExecutor>
```

Basic security XML syntax

"Basic" security involves coding the user registry and role authorization information in the server.xml file. It works well for development and test, but you'd never use it for a production setting.

The following is a sample basic security configuration we'll use to discuss the group access function:

```
<basicRegistry id="basic" realm="jbatch">
```

²⁸ The new batchGroupAdmin role is not illustrated in this picture, but the concepts are the same for that role.

```

<user name="ADMIN" password="adminpw" />
<user name="FRED" password="fredpw" />
<user name="MARY" password="marypw" />
<user name="JOHN" password="johnpw" />

  <group name="TEAM01">
    <member name="MARY" />
    <member name="JOHN" />
  </group>
</basicRegistry>

<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <user name="ADMIN" />
  </security-role>

  <security-role name="batchSubmitter">
    <user name="FRED" />
    <user name="MARY" />
  </security-role>

  <security-role name="batchGroupAdmin">
    <user name="JOHN" />
  </security-role>

  <security-role name="batchGroupMonitor">
    <user name="MARY" />
  </security-role>
</authorization-roles>

```

In this example we have four users defined: ADMIN, FRED, MARY, and JOHN.

We have one group defined: TEAM01. That group has two members: MARY and JOHN.

We have four roles defined:

batchAdmin	This role has one ID assigned: ADMIN
batchSubmitter	This role has two IDs assigned: FRED and MARY
batchGroupAdmin	This role has one ID assigned: JOHN
batchGroupMonitor	This role has one ID assigned: MARY

Imagine FRED submits a job under his ID. He has authority to do so because he has access to the batchSubmitter role. MARY also has batchSubmitter authority, but prior to this new function she would *not* be able to view Fred's jobs.

But notice that MARY *does* have access to the batchGroupMonitor role. And as already noted, MARY is a member of the TEAM01 group.

Therefore, if FRED submits a job and a group name of TEAM01 is associated with the job using the new <operationGroup> sub-element as illustrated previously, then MARY *would* be able to view the job submitted by FRED. She would have this authority because she is a member of the TEAM01 group *and* she has access to the batchGroupMonitor role.

The ID JOHN would have administrative authority over FRED's job because JOHN is a member of the TEAM01 group and he has access to the batchGroupAdmin role.

It's also possible to grant access to the *group*. For example:

```
<security-role name="batchGroupMonitor">
  <group name="TEAM01" />
</security-role>
```

That would provide `batchGroupMonitor` authority for all IDs that members of the `TEAM01` group.

SAF ID, GROUP, and the EJBROLE profile

In the previous section we saw the definition of the authorization roles in the `server.xml` file. That was the "basic" security model. You may use SAF to enforce role authorizations. This is done using the `EJBROLE` profile.

If you are using SAF as your user registry, then you would need to have the IDs defined to SAF. In our "basic security" example we had four IDs: `ADMIN`, `FRED`, `MARY`, and `JOHN`. Rather than define them in XML, you'd define them in SAF.

In our previous example we had one group: `TEAM01`. That would be defined in SAF, and to provide the same as illustrated in the "basic" example, we'd connect the `MARY` and `JOHN` IDs to the group.

Then define the two new `EJBROLE` profiles:²⁹

```
RDEFINE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin UACC(NONE)
RDEFINE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor UACC(NONE)
```

Then grant the IDs `READ` to the respective profiles:

```
PERMIT BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin
          CLASS(EJBROLE) ID(JOHN) ACCESS(READ)
PERMIT BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor
          CLASS(EJBROLE) ID(MARY) ACCESS(READ)
```

Or, grant the `TEAM01` group and allow the connected IDs to inherit:

```
PERMIT BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor
          CLASS(EJBROLE) ID(TEAM01) ACCESS(READ)
```

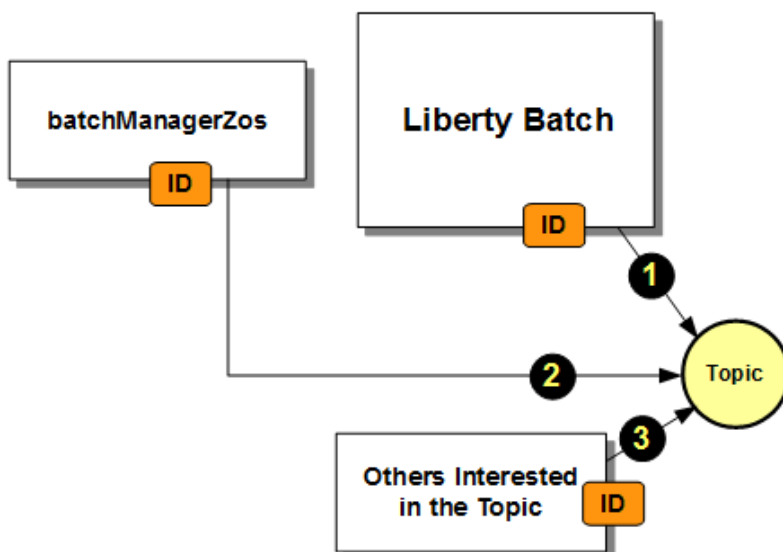
Securing the queueing environment – MQ

Note: The topic of "MQ Security" is broad and well understood by those with a responsibility to administer MQ. Rather than spell out all the details in this document – they are well-documented elsewhere – we will outline the key concepts and highlight where a focus on security is to be made. If you are using MQ, then consult with your MQ administrator

Single-server considerations

A single-server topology has no message queue for *job submission*. But it *may* make use of the pub/sub mechanism for monitoring the status of jobs. Consider the following picture:

²⁹ In this example we're showing the security prefix value as the default `BBGZDFLT`. You can create and use a different security prefix by defining your new prefix with the `APPL` class.

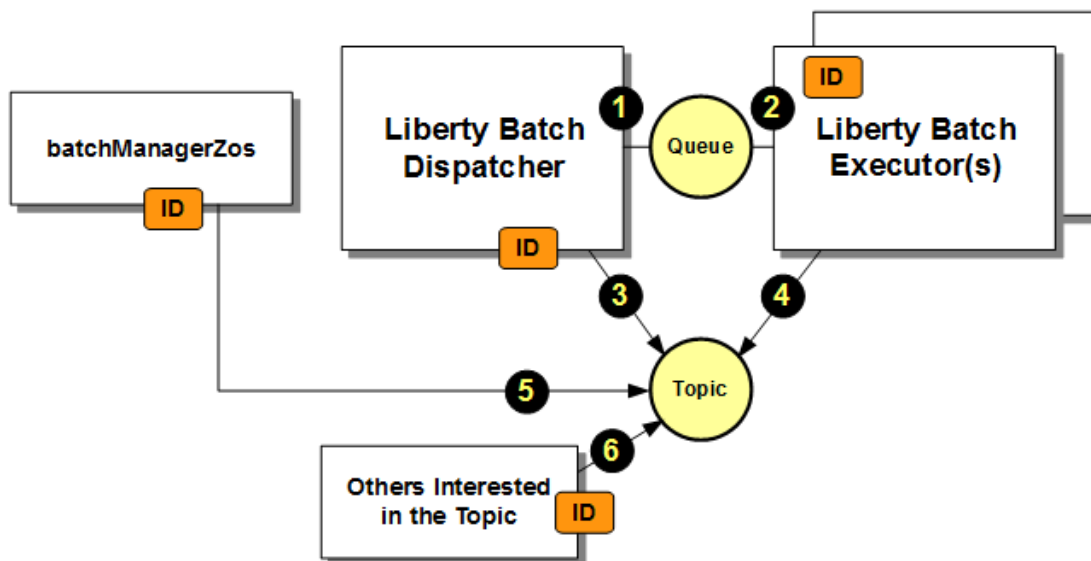
**Notes:**

1. The ID of the Liberty z/OS started task will need the authority to publish to the `/batch/#` topic space.
2. If you wish the job submission client – `batchManagerZos` at this time; the Java-based `batchManager` client does not at this time support subscribing to the topic – then the ID under which `batchManagerZos` runs must have authority to subscribe to the `/batch/#` topic space.
3. There may be other "interested parties" wishing to monitor the status of the jobs. The IDs under which these processes run would then require authority to subscribe to the `/batch/#` topic space.

When MQ is the platform on which the publish/subscribe pattern is implemented, then the focus is on securing the topic space. The Liberty server ID is what needs *publish* authority; and you grant *subscribe* authority to those clients that have an interest in monitoring the jobs.

Multi-server

This is an extension of the single-server model to include a PUT/GET queue that is used between the Dispatcher and the Executor:



Note: This assumes there are no jobs utilizing "job partitioning," which is the splitting of jobs into multiple "sub-jobs" that can run concurrently. See "Multi-server and job partitioning" on page 56 for more on this topic and how it affects the question of MQ security.

Notes:

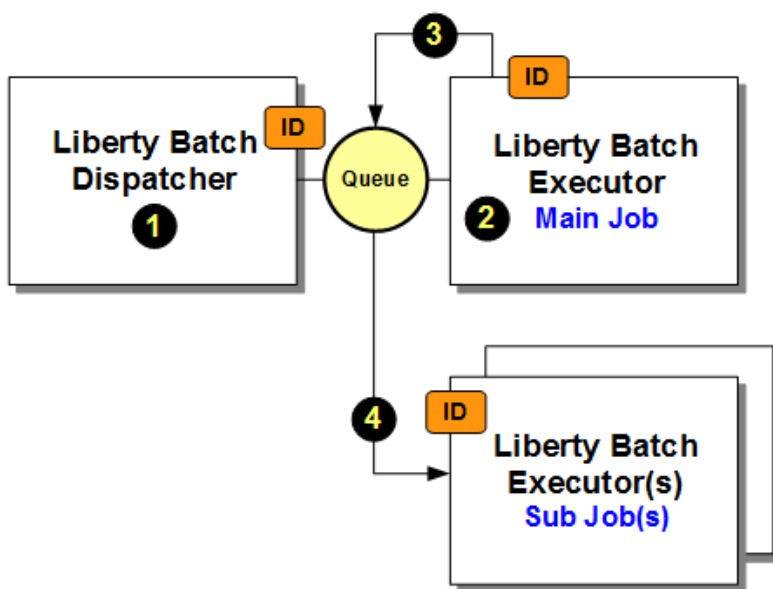
1. The ID of the Liberty z/OS server that serves as the Dispatcher will need authority to PUT messages to the job submission queue.
2. The ID (or IDs if multiple Executor servers are used) will need authority to GET messages from the job submission queue.
3. If you wish to make use of the pub/sub mechanism for monitoring jobs, then the ID of the Dispatcher will need the authority to publish to the `/batch/jobs/` topic space.
4. The ID of the Executor server(s) will also need authority to publish to the `/batch/#` topic space.
5. If you wish the `batchManagerZos` client to monitor the status of jobs using the events published to the topic space, then the ID under which it runs must have authority to subscribe to the topic space.
6. Finally, if there are other subscribers interested in monitoring the status of jobs, the IDs of those processes would need authority to subscribe to the `/batch/jobs/` topic space.

Multi-server and job partitioning

Job partitioning means a batch job is programmatically split into two or more sub-jobs and run concurrently. This can be done on multiple threads within a server, or across multiple server JVMs.

Note: It is possible to perform job partitioning within a *single server* environment. In that case the main job and the sub-jobs run in the same JVM, but on separate threads. The main job / sub-job coordination mechanism is then internal to the server, and involves no external message queueing. You may, however, be interested in using the *pub/sub* capability to monitor job activity within a single server with job partitioning. In that case the picture is what we showed under "Single-server considerations" on page 54.

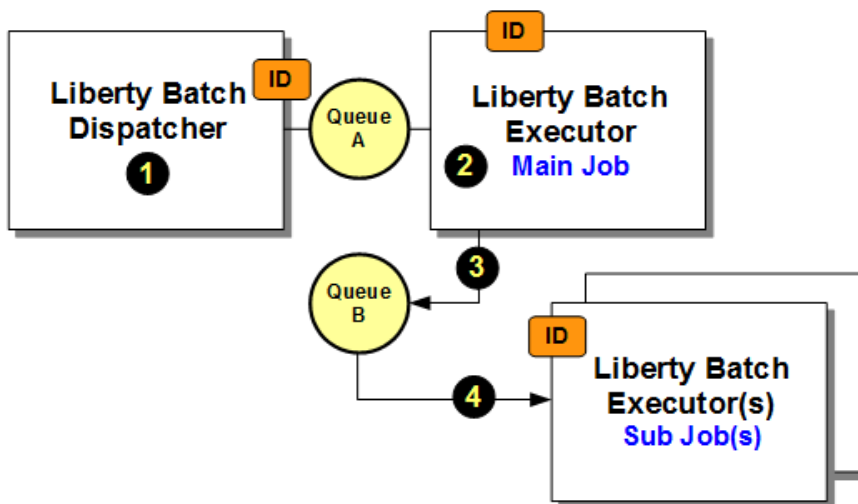
If you're running jobs partitions in a *multi-server environment*, then message queueing is the coordination mechanism between the "main job" and the "sub jobs". The picture we paint for this looks like this:



Notes:

1. The ID of the Liberty server acting at the Dispatcher needs authority to PUT a message on the queue
2. The ID of the Liberty server acting as the Executor of the main job needs authority to GET the message from the queue
3. The ID of the Liberty server acting as the Executor of the main job also needs authority to PUT a message on the queue
4. The ID of the Liberty server(s) acting as Executor of the sub-jobs needs authority to GET messages from the queue

A variation on this is the following picture, which employs a separate queue between the main job Executor and the sub-job Executor(s):



This is really just an extension of the earlier picture, with the main job Executor ID having GET authority to "Queue A" and PUT authority to "Queue B," and the sub-job Executor IDs having GET from "Queue B."

Miscellaneous and Reference Information

Liberty unauthenticated guest

The "unauthenticated guest" is the ID that is used prior to an authenticated ID getting control for a request. It is an ID that should have very limited authority. The Liberty z/OS default value is WSGUEST.

The following RACF sample commands illustrates how a group and the ID may be created:

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

Notes:

- The ID is connected to an isolated group, and should not be connected to any other groups where it may inherit authority the ID should not have.
- The NOPASSWORD and NOOIDCARD parameters prevent this ID from being used by anyone attempting to log onto the system.

If you wish to define a *different* unauthenticated guest value for an instance of Liberty, that is done with the following element of the server.xml file:

```
<safCredentials unauthenticatedUser="<your_value>" />
```

Typically you would see that value in this context:

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

Where:

- <safRegistry> indicates the use of SAF for the user registry
- <safAuthorization> indicates the use of SAF for application role checking (authorization)
- <safCredentials> defines the unauthenticated guest value and the profile prefix that is used on security profiles such as EJBROLE.

Elsewhere in this document you will see that block of XML used to define SAF for authentication and role checking. The profilePrefix value is shown elsewhere in this document. The unauthenticatedUser value is left uncoded, and is assumed to be the default WSGUEST.

Summary of updates: batchManagerZos and SAF

Available SERVER profiles:

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE) OWNER(SYS1)
SETROPTS RACLIST(SERVER) REFRESH
```

Granting of READ for WOLA access: and SAF access:

```

PERMIT BBG.ANGEL CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSCFM CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS (SERVER) ACCESS (READ) ID (<id>)

PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.SECPFX.BBGZDFLT CLASS (SERVER) ACCESS (READ) ID (<id>)

SETROPTS RACLIST (SERVER) REFRESH

```

Reminder: You may grant your server ID READ to the *other* SERVER profiles if you wish, or if you have a need for other function ... for instance, TXRRS because you are using JDBC Type 2 for DB2 access.

CBIND:

```

RDEFINE CBIND BBG.WOLA.LIBERTY.BATCH.MANAGER UACC (NONE) OWNER (SYS1)
PERMIT BBG.WOLA.LIBERTY.BATCH.MANAGER CLASS (CBIND) ACCESS (READ) ID (<id>)
SETROPTS RACLIST (CBIND) REFRESH

```

EJBROLE:

```

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)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin OWNER (SYS1)
UACC (NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor OWNER (SYS1)
UACC (NONE)
SETR RACLIST (EJBROLE) REFRESH

```

```

PE BBGZDFLT.com.ibm.ws.batch.batchAdmin CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchSubmitter CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchMonitor CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin CLASS (EJBROLE) ID (<id>)
ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor CLASS (EJBROLE) ID (<id>)
ACCESS (READ)
SETR RACLIST (EJBROLE) REFRESH

```

Configuration server.xml updates to support WOLA and SAF:

```

<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>zosLocalAdapters-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
</featureManager>

```

```

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

```

```
<zosLocalAdapters wolaGroup="LIBERTY"
  wolaName2="BATCH"
  wolaName3="MANAGER"/>
```

Summary of updates: batchManager with SAF registry and SAF role enforcement

Available SERVER profiles:

```
RDEFINE SERVER BBG.ANGEL UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC (NONE) OWNER (SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC (NONE) OWNER (SYS1)
SETROPTS RACLIST (SERVER) REFRESH
```

Granting of READ for SAF access:

```
PERMIT BBG.ANGEL CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS (SERVER) ACCESS (READ) ID (<id>)

PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS (SERVER) ACCESS (READ) ID (<id>)
PERMIT BBG.SECPFX.BBGZDFLT CLASS (SERVER) ACCESS (READ) ID (<id>)

SETROPTS RACLIST (SERVER) REFRESH
```

Reminder: You may grant your server ID READ to the other SERVER profiles if you wish, or if you have a need for other function ... for instance, TXRRS because you are using JDBC Type 2 for DB2 access.

EJBROLE:

```
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)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin OWNER (SYS1)
UACC (NONE)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor OWNER (SYS1)
UACC (NONE)
SETR RACLIST (EJBROLE) REFRESH

PE BBGZDFLT.com.ibm.ws.management.security.resource.allAuthenticatedUsers
  CLASS (EJBROLE) ID (<id>) ACCESS (READ)
PE BBGZDFLT.com.ibm.ws.batch.batchAdmin CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchSubmitter CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchMonitor CLASS (EJBROLE) ID (<id>) ACCESS (READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin CLASS (EJBROLE) ID (<id>)
ACCESS (READ)
```

-or-

```
PE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor CLASS(EJBROLE) ID(<id>)
ACCESS(READ)
SETR RACLIST(EJBROLE) REFRESH
```

Configuration server.xml updates to support SAF:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
</featureManager>

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

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

Summary of updates: batchManager, SAF authentication and roles, SAF SSL**Available SERVER profiles:**

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) OWNER(SYS1)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE) OWNER(SYS1)
SETROPTS RACLIST(SERVER) REFRESH
```

Granting of READ for SAF access:

```
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(<id>)

PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(<id>)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(<id>)

SETROPTS RACLIST(SERVER) REFRESH
```

Reminder: You may grant your server ID READ to the other SERVER profiles if you wish, or if you have a need for other function ... for instance, TXRRS because you are using JDBC Type 2 for DB2 access.

EJBROLE:

```
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)
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin OWNER(SYS1)
UACC(NONE)
```



```
RDEF EJBROLE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor OWNER(SYS1)
UACC (NONE)
SETR RACLIST(EJBROLE) REFRESH
```

```
PE BBGZDFLT.com.ibm.ws.management.security.resource.allAuthenticatedUsers
  CLASS(EJBROLE) ID(<id>) ACCESS(READ)
PE BBGZDFLT.com.ibm.ws.batch.batchAdmin CLASS(EJBROLE) ID(<id>) ACCESS(READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchSubmitter CLASS(EJBROLE) ID(<id>) ACCESS(READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchMonitor CLASS(EJBROLE) ID(<id>) ACCESS(READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchGroupAdmin CLASS(EJBROLE) ID(<id>)
ACCESS(READ)
-or-
PE BBGZDFLT.com.ibm.ws.batch.batchGroupMonitor CLASS(EJBROLE) ID(<id>)
ACCESS(READ)
SETR RACLIST(EJBROLE) REFRESH
```

Configuration server.xml updates to support SAF:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosSecurity-1.0</feature>
  <feature>ssl-1.0</feature>
</featureManager>

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

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

Creation of CA cert, signed server cert, and keyrings:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty') OU('LIBERTY'))
  WITHLABEL('LibertyCA.LIBERTY') TRUST SIZE(2048) NOTAFTER(DATE(2018/12/31))
RACDCERT ID(<server_id> GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com')
  O('IBM') OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY')
  SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY')) SIZE(2048)
  NOTAFTER(DATE(2018/12/30))
RACDCERT ID(<server_id> ADDRING(Keyring.LIBERTY)
RACDCERT CONNECT(ID(<server_id> LABEL('DefaultCert.LIBERTY')
  RING(Keyring.LIBERTY)) ID(<server_id>)
RACDCERT CONNECT(CERTAUTH LABEL('LibertyCA.LIBERTY'))
```

```
RING(Keyring.LIBERTY) ID(<server_id>)
SETR RACLIST(DIGTCERT DIGTRING) REFRESH
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(<server_id>) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(<server_id>) ACCESS(READ)
SETR RACLIST(FACILITY) REFRESH
```

Creation of client keyring and connecting CA certificate

```
RACDCERT ID(FRED) ADDRING(Keyring.FRED)
  RACDCERT CONNECT(CERTAUTH LABEL('LibertyCA.LIBERTY'))
  RING(Keyring.FRED) ID(FRED)
SETR RACLIST(DIGTCERT DIGTRING) REFRESH
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(FRED) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(FRED) ACCESS(READ)
SETR RACLIST(FACILITY) REFRESH
```

Document Change History

Check the date in the footer of the document for the version of the document.

<i>February 22, 2017</i>	Original document at time of Techdoc creation
<i>February 24, 2017</i>	Updated document with the assigned WP number and republished.
<i>March 19, 2018</i>	Updated with information on the new batch group authorization function provided with 18.0.0.1
<i>September 6, 2018</i>	Updated a note on the Admin Center and its ability to purge jobs.

End of WP102696