z/OS Connect Enterprise Edition V3.0

Getting Started Guide

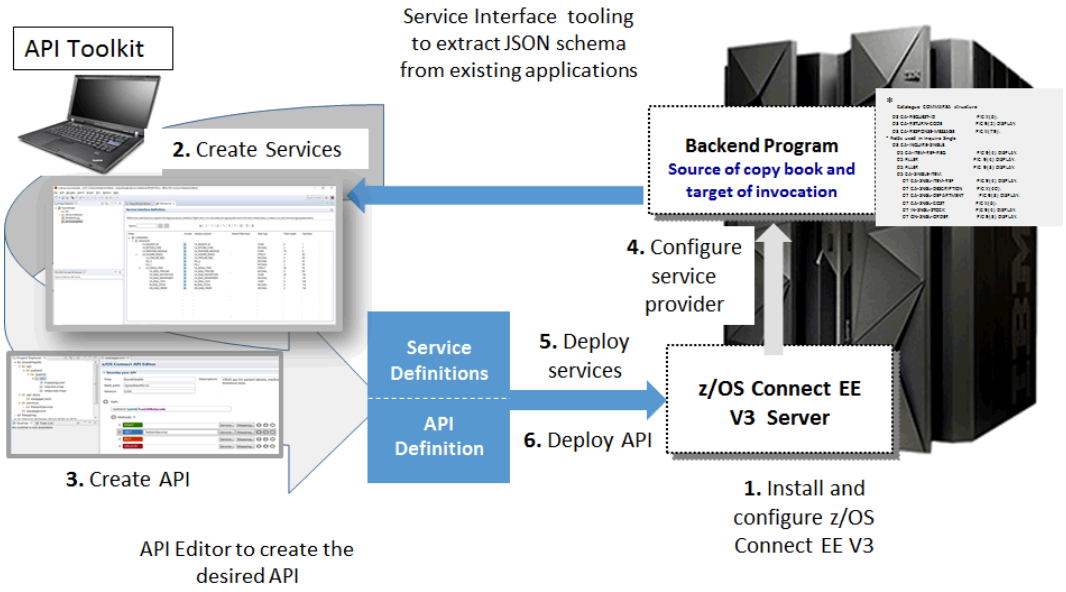
for CICS, IMS, Db2 and MQ

Version Date: July 24, 2020

(If you have comments or feedback on the contents of this document, please send an e-mail to
Mitch Johnson (mitchj@us.ibm.com).

Table of Contents

IBM® z/OS® Connect Enterprise Edition V3.0 provides a framework that enables z/OS based programs and data to participate fully in the new API economy for mobile and cloud applications.



IBM z/OS Connect Enterprise Edition V3.0 (zCEE) provides RESTful API access to z/OS subsystems, such as CICS®, IMS™, IBM® MQ, Db2®, as well as potentially other z/OS applications. The framework provides concurrent access, through a common interface, to multiple z/OS subsystems. In addition, z/OS Connect EE.4 and later provides support for outbound RESTful API from CICS, IMS and other MVS applications. This rich framework also provides a common security model, as well as logging, tracking and API development and deployment services.

The goal of this document is to provide a step-by-step guide to setting up z/OS Connect EE servers for usage with either CICS, IMS, MQ or Db2. Emphasis will be placed on CICS, IMS, Db2 and MQ since they are most common use cases.

Document Overview

This document will provide a task-oriented outline for getting started with z/OS Connect Enterprise Edition (zCEE) V3.0. The document is organized in the following way:

Program numbers and FMIDs

Program number: **5655-CE3** **z/OS Connect EE V3.0 continuous delivery**
Base FMID: **HZC3000** **z/OS Connect EE V3.0 core product**
FMID: **JZC3002** **z/OS Connect EE optional CICS dependencies**

Program number: **5655-CE5**
Base FMID: **HZC3000** **z/OS Connect EE V3.0 core product**
FMID: **JZC3002** **z/OS Connect EE optional CICS dependencies**
FMID: **JZC3003** **z/OS Connect EE unlimited activation**

Recommended Maintenance

Current release information for both the server and the API toolkit can be found at this location:

https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/com.ibm.zosconnect.doc/overview/change_history.html

The IBM Knowledge Center

This document leverages the content found in the IBM Knowledge Center for IBM z/OS Connect EE, which is found at this location:

https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/com.ibm.zosconnect.doc/welcome/WelcomePage.html

IBM Techdocs

This document, as well as other collateral related to IBM z/OS Connect EE, can be found at the following Techdoc location:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102724

IBM z/OS Connect Enterprise Edition V3.0 web page

Here is the URL for the product web page:

http://www.ibm.com/software/products/en/zos-connect-enterprise-edition

IBM developerWorks Developer web pages

Here is the URL for the developerWorks Overview of the z/OS Connect EE web page:

https://developer.ibm.com/mainframe/products/zosconnect/

Here is the URL for a developerWorks article which describes the MQ Service Provider for z/OS Connect:
https://www.ibm.com/developerworks/community/blogs/messaging/entry/The_MQ_Service_Provider_for_z_OS_Connect

IBM Support web pages

Here is the URL for a description of what is new with each z/OS Connect EE refresh

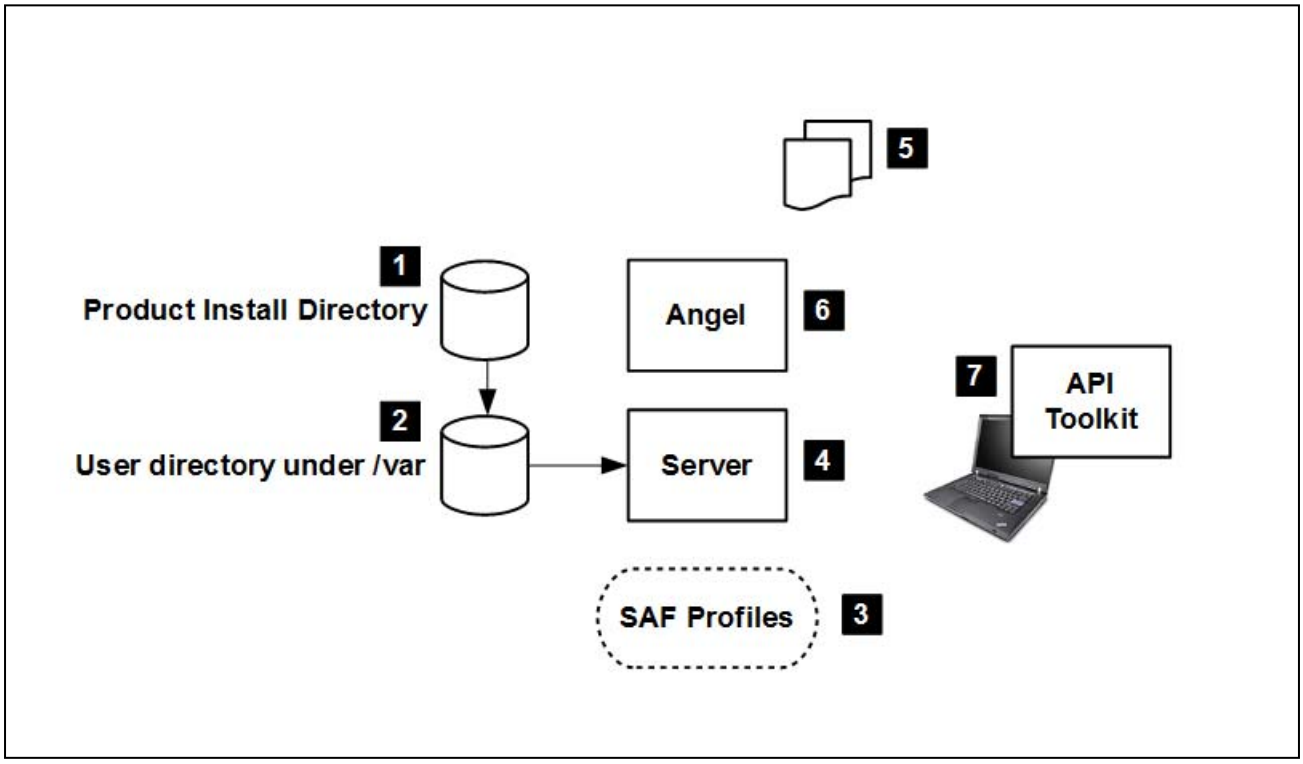
https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/com.ibm.zosconnect.doc/overview/change_history.html

Here is the URL for information for upgrading Liberty profile for z/OS Connect EE:

https://www-01.ibm.com/support/docview.wss?uid=swg21993579

Installation and Initial Setup

Picture overview of the steps in this section



Notes:

1. SMP/E is used to install z/OS Connect EE. This is a standard SMP/E install process. The result is a file system mounted at the location you specify and SMP/E target data sets.
2. The *zconsetup* shell script must be executed to create a set of directories under directory */var/zosconnect* where a z/OS Connect EE *extensions* directory is located. The *extensions* directory will contain properties files which need to available when starting a z/OS Connect EE server.
3. SAF profiles are required to allow z/OS Connect EE to operate as a started task and performed authorized functions.
4. Create a basic Liberty server with the z/OS Connect EE feature using a simple shell script.
5. Copy the sample JCL procedures to your procedure library from the SBAQSAMP TLIB.
6. The Angel process is only required in some circumstances, and there may already be an Angel present on your system which may be use for z/OS Connect. We will guide you through the process of determining the Angel to use – existing, or new.
7. Install the z/OS Connect EE API Tool Kit on your workstation.

SMP/E install of z/OS Connect EE and post install customization

IBM z/OS Connect EE (zCEE) is installed using SMP/E. This will require someone with SMP/E skills to accomplish this.

The Knowledge Center page for installing IBM z/OS Connect EE is here:

https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/com.ibm.zosconnect.doc/installing/smpe.html

Do the following:

- Perform the SMP/E steps indicated in the program directory to install the product.
- Create directory */var/zosconnect* and mount an HFS filesystem at this mount point. An HFS filesystem is sufficient for this purpose since updates to the */var/zosconnect* directory are small and are only done during the initial install of z/OS Connect and when service providers not shipped with z/OS Connect are added to the base product
- Create a mount point named *servers* in */var/zosconnect* and mount a ZFS filesystem at this mount point. Ensure the identities that will be used to create and under the server will run have write access to this directory.

Tech Tip: We are recommending that a dedicated filesystem for */var/zosconnect* be created and mounted for each LPAR. This is done so the configuration information is not lost when the root filesystem on a LPAR is updated with a refresh of z/OS.

Tech Tip: The directory will be the default location for server configuration files and application artifacts. A ZFS filesystem is being used to allow for growth.

Tech Tip: The runtime uses environment variable **WLP_USER_DIR** to determine the location of server configuration files and application artifacts. If no value is provided for **WLP_USER_DIR**, the default value is */var/zosconnect*. If a value other than the default will be used for **WLP_USER_DIR**, then mount a ZFS file system at this directory. For example, if **WLP_USER_DIR** is set to */var/ats/zosconnect*, create a ZFS filesystem and mount the ZFS filesystem at */var/ats/zosconnect/*.

```
MOUNT FILESYSTEM('OMVS.ATS.ZCEE.ZFS') TYPE(ZFS)
 MODE(RDWR)MOUNTPOINT('/var/ats/zosconnect')
```

□ Verify the product installation file system is mounted R/W. The *zconsetup* script executed next will need to create a symbolic link from this file system to directory */var/zosconnect/V3R0/extensions* and the installation filesystem needs to be R/W for this to succeed.

Tech Tip: The *zconsetup* script creates a symbolic link from the */wlp/etc/extensions* sub directory embedded with z/OS Connect product directory structure to external directory */var/zosconnect/v3r0/extensions*. The former directory is usually mounted read/only while the latter is mounted read/write. This allows the customization for additional product service providers on an LPAR by LPAR basis. We also recommend that the *zconsetup* script be run in the SMP/E maintained filesystem, so the symbolic link is not lost when service is applied, and the z/OS Connect filesystem is refreshed.

□ Use the TSO *OMVS* command or use Telnet or SSH to open an OMVS shell and go to directory */usr/lpp/IBM/zosconnect/v3r0/bin*[1]. Log on with or switch to an ID that has authority to create a symbolic link and to create directories.

□ Run the script with this command: ***./zconsetup install*** to create a symbolic link between the product directory and this *extensions* directory.

□ Remount the product installation file system as R/O.

Review the file system. You should see a directory structure like this:

Directory	*Purpose*
`/usr/lpp/IBM/zosconnect/v3r0/bin`	`Product Code`
`/usr/lpp/IBM/zosconnect/v3r0/dev/`	`Java classes for user service providers`
`/usr/lpp/IBM/zosconnect/v3r0/doc`	`Java Doc zip file`
`/usr/lpp/IBM/zosconnect/v3r0/imsmobile`	`IMS Service Provider`
`/usr/lpp/IBM/zosconnect/v3r0/runtime/lib/`	`Feature Files`
`/usr/lpp/IBM/zosconnect/v3r0/wlp`	`WebSphere Liberty product code`
`/usr/lpp/IBM/zosconnect/v3r0/wlp/etc/extensions`	`Contains symbolic link to directory /var/zosconnect/extensions`
`/usr/lpp/IBM/zosconnect/v3r0/zconnbt.zip`	`z/OS Connect EE build tool`
`/var/zosconnect/v3r0/extensions`	`Properties files for product features`[2]
`/var/zosconnect/servers`	`Server configuration files and applications`[3]

[1] This document assumes z/OS Connect EE V3 was installed into the default directory.

2 This subdirectory contains "property" files which identify which products (i.e. IBM MQ) have been added to z/OS Connect to extend its functionality. This directory name should not be changed.

3 The value for this directory is based on environment variable WLP_USER_DIR. The default value is shown.

Essential prerequisites

You will need the following:

- z/OS 2.1 or higher
- IBM 64-bit SDK for z/OS, Java Technology Edition V8.0.0 or higher

Do the following:

□ Verify your level of z/OS is 2.1 or higher

□ Check to see if you have a valid 64-bit IBM Java SDK for z/OS, V8.0.0 instance. If not available have your system administrator installed V8.0.0.

Liberty Angels

Some features of z/OS Connect EE will require that a Liberty Angel be active[4] [5].

You may already have an Angel if you have z/OSMF or other Liberty instances started[6]. If that is the case, that Angel *may* be able to be used for your z/OS Connect EE servers.

If you do use an existing Angel process, ensure that it is compatible with z/OS Connect EE. If you see message: *CWWKB0307E: The angel process on this system is not compatible with the local communication service,* this means the existing Angel is back leveled with the requirements of z/OS Connect and needs to be upgraded. Consider using the Angel code shipped with z/OS Connect EE by configuring the Angel JCL start procedure to point to the WebSphere Liberty Profile (WLP) directories shipped with z/OS Connect EE and providing a unique name to be used for z/OS Connect Liberty servers.

Named Angels

Each Angel can be uniquely identified by a name at startup. An Angel with no name provided at startup is known as the default Angel.

All Liberty servers (including a z/OS Connect server) can be configured to select which Angel it will use for authentication by specifying a system property. If no Angel name is specified by a Liberty server then the default Angel (i.e. one with no name) will be selected. Another system property can be set to require the successful connection to Angel to continue the startup of the server. That is, if an Angel is not available the Liberty server will shut itself down.

- To provide these properties for a z/OS Connect EE server:

1. Create an options file for angel properties, e.g. *angel.options* in an OMVS directory, e.g. */var/zosconnect* and enter the system properties as below:

```
-Dcom.ibm.ws.zos.core.angelName=AngelName
-Dcom.ibm.ws.zos.core.angelRequired=true
```

4 Most notably, WOLA for access into CICS and/or security. Other functions may as well. It is best to anticipate and have the Angel present for those cases where it is needed.

5 For more on Liberty z/OS and the Angel process: http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110

6 z/OSMF 2.1 is based on Liberty z/OS, and it requires the Angel for access to z/OS authorized services.

(Where *AngelName* is the name of the Angel to be used for security)

2. Use the *JAVA_OPTIONS* environment variable and provide these properties in this file using the STDENV input in the JCL procedure, see page 18. The STDENV DD statement can reference a file in an OMVS directory.

```
_BPX_SHAREAS=YES
JAVA_HOME=<Java home directory>
#JVM_OPTIONS=<Optional JVM parameters>
WLP_USER_DIR=/var/zosconnect
JVM_OPTIONS=-Xoptionsfile=/var/zosconnect/angel.options                -
```

Tech Tip: Generic SERVER profiles for controlling access to Angels should be avoided. The presence of a generic Angel resource may have unintended consequences regarding access to privileged functions.

Please note that if named Angels are used then additional SERVER SAF profiles will need to be defined and permission granted to the SAF identities of the z/OS Connect EE servers, see page 17.

For example, if Angel is started with a name of PRODUCTION, then a SAF SERVER profile for this name, i.e., BBG.ANGEL.PRODUCTION must be defined and the z/OS Connect EE server running under identity USER1 must be given READ access, e.g.:

```
RDEFINE SERVER BBG.ANGEL.PRODUCTION UACC(NONE) OWNER(SYS1)
PERMIT  BBG.ANGEL.PRODUCTION CLASS(SERVER) ACCESS(READ) ID(USER1)
```

Note: For the *initial* setup we will keep things simple and host some elements of the security model in the server's server.xml file. To understand how to move beyond these simple security definitions, see *Beyond the simple server.xml security elements* on page 76. What follows are z/OS security elements that must be in place before operating the z/OS Connect EE server.

SAF Resources

The SAF resources for z/OS Connect EE are best planned and created ahead of time. This will best utilize your time working coordinating with the security administrator.

Group and Server IDs

Note: If you already have an Angel process in place, you do *not* need to create another Angel ID. You simply make use of the existing Angel and its ID. Also, it is not required that the Liberty IDs be connected to a common group. We illustrate that here as one approach.

Work with your security administrator and do the following:

- Plan the values you will use for your Angel ID and server ID, and the group ID.
- Use the following examples as guides and create the group and IDs:

Creates a Liberty Profile group ID

```
ADDGROUP libGroup OMVS(AUTOGID) OWNER(SYS1)
```

Creates the Angel ID and connects it to the Liberty Profile group

Tech Tip: The combination of NOPASSWORD and NOIDCARD makes this a PROTECTED identity. This means that this identity cannot be used to access this system by any means that requires a password to be specified, such as a TSO logon, CICS sign on, or via a batch job that specifies a password on the JOB statement. These attributes also mean that this identity will not be revoked if an attempt is made to access the system with an invalid password.

```
ADDUSER angelID DFLTGRP(libGroup) OMVS(AUTOUID
                HOME(angel_home) PROGRAM(/bin/sh)) NAME('Liberty Angel')
        OWNER(libGroup) NOPASSWORD NOOIDCARD
```

Creates the Liberty Profile server ID and connects it to the Liberty Profile group

```
ADDUSER libertyID DFLTGRP(libGroup) OMVS(AUTOUID
                HOME(server_home) PROGRAM(/bin/sh)) NAME('Liberty Server')
        OWNER(libGroup) NOPASSWORD NOOIDCARD
```

OMVS and Surrogate Permissions

Tech Tip: An alternative to using surrogate access to create the server and its directory structure is the using the OMVS ***chown*** command to change the directory and file ownership of an existing server's configuration to the server's identity and group, e.g.

```
cd /var/zosconnect/servers
chown libertyID:libGroup  serverName
chown -R libertyID:libGroup serverName
```

A common issue during the configuration of a z/OS Connect EE server is caused by improper or incorrect file permission bit and ownership settings. Most of these can be addressed if the same RACF identity that will be used by the started task is also used to perform the initial configuration of the server. Since the identities associated with started task are normally restricted and cannot be used for accessing TSO or OMVS shells the alternative is to use RACF surrogate access. That allows an administrative user the ability to invoke commands and perform functions using the same identity as will be used for the z/OS Connect EE server started task.

 Use the following examples as guides and create the surrogate resources and permit access:

```
Define a SURROGAT profile for the z/OS Connect EE server identity
RDEFINE SURROGAT BPX.SRV. libertyID
Define a SURROGAT profile to allow job submission as the identity.id
RDEFINE SURROGAT  libertyID.SUBMIT
Permit an administrative identity to act as a surrogate of the Liberty task identity
PERMIT BPX.SRV. libertyID CLASS(SURROGAT) ID(adminUser) ACC(READ)
PERMIT libertyID.SUBMIT CLASS(SURROGAT) ID(adminUser) ACC(READ
```

SETROPTS RACLIST(SURROGAT) REFRESH

These commands allow the *adminUser* to use the OMVS switch user command with the *-s* flag (e.g. ***su -s***) to switch identities to the Liberty's started task identity and invoke OMVS commands (when creating configuration files and directories as the Liberty's started task identity, this ensures all permission bits are set properly). Access to the SUBMIT resource allows an *adminUser* to submit jobs as the Liberty's servers task identity without providing the started task user's password (see the JCL in *Creating a server* on page 80 as an example).

Tech Tip: Optionally assign the Liberty server identity a password.

ALTUSER libertyID PASSWORD(password) NOEXPIRED

Assigning a password to this identity provides an advantage to an administrator when using FTP to install application artifacts(e.g., API archives, service archives, API requester archives). Authenticating with server's identity ensures proper file ownership when using FTP to install API these artifacts. Note adding a password does disable the PROTECTED user attribute.

STARTED profiles

The STARTED profiles are used to assign the identity when the server is started as a z/OS started task. They are based on the JCL start procedure name. z/OS Connect EE comes with sample JCL, and you may keep the default JCL procedure names or create your own.

Note: if you already have an Angel process in place, you do *not* need to create a new JCL procedure or STARTED profile. You simply make use of the existing Angel JCL procedure and its authorization ID.

Work with your security administrator and do the following:

- ☐ Plan your JCL start procedure names (either default or your own values)
- ☐ Use the following examples as guides and create the STARTED profiles:

Creates the STARTED profile for the Angel Process

```
RDEF STARTED angelProc.* UACC(NONE) STDATA(USER(angelID)
            GROUP(libGroup) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

Creates the STARTED profile for the Liberty Profile server

```
RDEF STARTED serverProc.* UACC(NONE) STDATA(USER(libertyID)
            GROUP(libGroup) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

Refreshes the STARTED class profiles

```
SETROPTS RACLIST(STARTED) REFRESH
```

SERVER profiles

The SERVER profiles grant access to authorized services z/OS Connect EE may need. Some of the SERVER profiles are not strictly required for z/OS Connect EE, but you may decide to create all the profiles indicated just to have them on hand in case you need them later. See the notes that follow for a brief explanation of which are optional and why.

Work with your security administrator and do the following[7]:

- ☐ Use the following examples as guides and create the SERVER profiles:

Grants an ID general access to the Angel process for authorized services

```
RDEF SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(libertyID)
```

Grants an ID general access to a named Angel process for authorized services

```
RDEF SERVER BBG.ANGEL.angelName UACC(NONE) OWNER(SYS1)
PERMIT BBG.ANGEL.angelName CLASS(SERVER) ACCESS(READ) ID(libertyID)
```

Controls which server processes can use the BBGZSAFM authorized module in the Angel process

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(libertyID)
```

Controls which server processes can use BBGZSAFM for SAF authorization services

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

7 These SERVER profiles can be used by *any* Liberty z/OS, whether z/OS Connect EE or not. You may already have these profiles created. If so, then you do *not* need to create the profile, you need only grant your server ID READ to the profile.

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

Controls which server processes can use BBGZSAFM for WLM services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSWLM CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Controls which server processes can use BBGZSAFM for RRS services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.TXRRS CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Controls which server processes can use BBGZSAFM for z/OS Dump services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSDUMP CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Controls which server processes can use BBGZSAFM for WOLA services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Controls which server processes can use BBGZSAFM for LOCALCOM services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE)
OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Controls which server processes can use the authorized client module BBGZSCFM

```
RDEF SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSCFM CLASS(SERVER) ACCESS(READ) ID(libertyID)
```

Controls which server processes can use optimized local adapter client services

```
RDEF SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Controls access to EJBROLE definitions based on the prefix in use for a server

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

Controls access to IFAUSAGE services (SMF) based on the prefix in use for a server

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Writing SMF records also requires access to this FACILITY resource

```
RDEF FACILITY BPX.SMF UACC(NONE) OWNER(SYS1)
PERMIT BPX.SMF CLASS(FACILITY) ACCESS(READ) ID(libertyID)
```

Controls access to AsyncIO services based on the prefix in use for a server

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSAIO CLASS(SERVER) ACCESS(READ)
ID(libertyID)
```

Refreshes the SERVER class profiles

```
SETROPTS RACLIST(SERVER,FACILITY) REFRESH
```

Notes:

- SAFCRED – needed if you intend to use SAF for security elements such as registry, certificates and EJBROLEs. For initial validation you do not need this, but for any real-world usage of z/OS Connect EE you will need this service available.
- ZOSWLM – needed if you wish to classify work using WLM. Initially you won't do this, but later you might. Better to create now and have available when you need it.
- TXRRS – needed for access to RRS for transaction coordination. You should not need this for z/OS Connect EE as it does not create global transactions and therefore does not need the services of RRS for that purpose. You may want to create and have on hand for *other* Liberty servers not running z/OS Connect EE.
- ZOSDUMP – needed if you wish to use the MODIFY interface to the Liberty z/OS server to process a dump operation. This is good to have available if IBM support requests a dump for your z/OS Connect EE server.
- PRODMGR – needed if you wish to enable IFAUSAGE (SMF) for Liberty on z/OS.
- ZOSAIO – needed if you wish to permit the enablement of the use of Asynchronous TCP/IP sockets I/O for Liberty on z/OS.
- LOCALCOM – needed for optimized local adapter services.
- WOLA – needed if you wish to use WebSphere Optimized Local Adapter support for cross memory communications between tasks.

With z/OS Connect EE installed and the required SAF profiles in place, you are ready to create your server and perform initial validation of the environment.

z/OS Connect Server creation

The Knowledge Center URL for this task is:

https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/com.ibm.zosconnect.doc/configuring/creating_zC_server.html

Do the following:

- Open a Telnet, SSH or OMVS session to your z/OS system. *Log in as or switch to the ID you planned to use for the server's started task.*
- *Export the JAVA_HOME environment variable:*

  export JAVA_HOME=path_to_your_64-bit_Java_SDK

Tech Tip: Add this *export* command to the *.profile* file located in the user's home directory.

- Test to make sure the ID can instantiate a JVM. Do the following:
  - Change directories to the **/bin** directory of your JAVA_HOME location
  - Issue the command: ***./java –version***
  - You should receive something like this:

```
java version "1.8.0"
Java(TM) SE Runtime Environment (build pmz6480sr3fp20-20161019_02(SR3 FP20))
IBM J9 VM (build 2.8, JRE 1.8.0 z/OS s390x-64 Compressed References
20161013_322
271 (JIT enabled, AOT enabled)
J9VM - R28_Java8_SR3_20161013_1635_B322271
JIT  - tr.r14.java.green_20161011_125790
GC   - R28_Java8_SR3_20161013_1635_B322271_CMPRSS
J9CL - 20161013_322271)
JCL - 20161018_01 based on Oracle jdk8u111-b14
```

Note: If it fails (with an error JVMJ9VM011W EDC5204E) then it is likely because your ID does not can get the memory needed to create the JVM. Adjust[8] the size parameters of the user's TSO segment using the TSO *ALTUSER* command:

ALU user-name TSO(SIZE(1048576)) OMVS(ASSIZEMAX(1073741824) MEMLIMIT(1G))

When you have successfully checked the version of Java, then proceed.

 Go to the *bin* directory where z/OS Connect EE is installed, e.g.

/usr/lpp/IBM/zosconnect/v3r0/bin.

 Export environment variable *WLP_USER_DIR* to identify the directory location of where the server configuration will be created.

export WLP_USER_DIR=/var/zosconnect

8 You may need to work with your system administrator to accomplish this. The key point the ID must be able to instantiate a JVM or you cannot proceed. This test checks to see if the ID has the ability. If not, correct the issue.

 Use the command ***zosconnect create serverName --template=templateName*** to create a server:

Where ***templateName*** can be:

- *zosconnect:apiRequester* for an API requester enabled z/OS Connect server
- *zosconnect:default* template for base/OS Connect servers
- *zosconnect:sampleCicsIpicCatalogManager* for a sample CICS enabled z/OS Connect server
- *zosconnect:sampleDb2Project* for a sample Db2 enabled z/OS Connect server
- *zosconnect:samplePhonebook* for a sample IMS enabled z/OS Connect server
- Where ***serverName*** is any value you wish, such as server1. Capture the name of your server here:

Tech Tip: The differences between these templates are the features added to the *featureManager* configuration element in the initial server.xml, e.g. the IMS template adds the *imsmobile* feature the CICS template add the *cicsService* feature and the API requester template adds the *apiRequeste*r feature. Perhaps more information are the directories created in the *../resources/zosconnect* subdirectory in the server's configuration path. All the templates create the *apis, services* and *rules* subdirectories but only the *apiRequesters* templates creates the *apiRequesters* subdirectory. If the apiRequesters feature is added to an existing server be sure to manually create this subdirectory with the correct permission bits and ownership.

 Go to *the /var/zosconnect/servers* (i.e. the directory specified by environment variable *WLP_USER_DIR)* directory and verify that a sub-directory with the name ***serverName*** was created, and under the ***serverName*** directory there exists a server.xml file.

N. B. For an example of the JCL that could be used to create a server see section "Creating a server" on page 80.

Tech Tip: Consider configuring a RACF SURROGAT resource as describe earlier that would allow you to switch to the authorization identity being used by the server in an OMVS shell prompt. For example, the commands below will allow USER1 to use the OMVS command ***su –s libserv*** to switch identity from *user1* to *libserv* and any directories or files created will be owned by *libserv*.

```
RDEFINE SURROGAT BPX.SRV.LIBSERV
PERMIT BPX.SRV.LIBSERV CLASS(SURROGAT) ID(USER1) ACC(READ)
```

Tech Tip: The same value used for WLP_USER_DIR when creating the server needs to be exported in the JCL used to start the server.

TCP ports and host element

A few minor updates to */var/zosconnect/servers/**serverName**/server.xml* may be required at this point.

> **Tech Tip:** Use the Ascii editor available when using ISPF option 3.4 or 3.17 when accessing OMVS directories.

Do the following:

□ Consult with your TCP networking administrator and see if the default ports of 9080 and 9443 are acceptable. If not, plan the two TCP ports you will use:

□ Edit the server.xml file and update the ports specified in the httpEndPoint element.

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

The two ports should reflect either the default values (shown) or your planned values.

□ Save the file.

Start a z/OS Connect EE server

Earlier you created the STARTED profiles to assign an identity to the started task. z/OS Connect EE comes with sample JCL start procedures you can copy to your PROCLIB and customize to your environment.

Do the following:

- Copy the sample server JCL from member BAQSTRT in your SMP/E SBAQSAMP target library to your PROCLIB. Make sure the resulting procedure's JCL does not have 'numbers' off to the right of the member. If you find them, issue command *unnum* to remove the numbers. That will also set the ISPF profile to NUMBER OFF.
- Rename the procedure so it matches the STARTED profile you created for the server.
- Customize the server JCL:

```
//BAQSTRT  PROC PARMS='defaultServer'
//*
   (comment lines removed to save space in this document)
//*-------------------------------------------------------------------
//* Start the Liberty server
//*
//* STDOUT  - Destination for stdout (System.out)
//* STDERR  - Destination for stderr (System.err)
//* STDENV  - Initial z/OS UNIX environment for the specific
//*           server being started
//*
// SET ZCONHOME='<Install path>'  1
//*
//ZCON     EXEC PGM=BPXBATSL,REGION=0M,MEMLIMIT=4G,
//         PARM='PGM &ZCONHOME./bin/zosconnect run &PARMS.'
//STDOUT   DD   SYSOUT=*
//STDERR   DD   SYSOUT=*
//STDIN    DD   DUMMY
//STDENV   DD   *
_BPX_SHAREAS=YES
_CEE_RUNOPTS=HEAPPOOLS(ON),HEAPPOOLS64(ON)
JAVA_HOME=<Java home directory>   2
WLP_USER_DIR=<User directory>     3
#JVM_OPTIONS=<Optional JVM parameters>
//*
// PEND
//
```

Notes:

1. Set the *<Install path>* value to the path of the z/OS Connect EE install location, e.g. */usr/lpp/IBM/zosconnect/v3r0* or whatever your value is. Make sure to enclose the value in single quotes as shown in the JCL.
2. Set JAVA_HOME= to the path to your 64-bit IBM Java SDK, e.g. */usr/lpp/java/J8.0_64*

Tech Tip: The same value used for WLP_USER_DIR used when creating the server needs to be exported in the JCL used to start the server.

3. Set WLP_USER_DIR to the location where the shared resources and server definitions will be created. The default value is */var/zosconnect*.

If you intend to use an already-existing Angel process, then skip over the following steps[9]. Otherwise,

> **Tech Tip:** The name of the Angel can be provided using the NAME parameter on the start command, e.g. ***S BAQZANGL,NAME=PRODUCTION***. Any Liberty server that will use this Angel for security must be configured as described above using the *com.ibm.ws.zos.core.angelName* and *com.ibm.ws.zos.angelRequired* system properties, see *Named Angels* on page 13. Using a named Angel will also require additional RACF resources, see section

follow these steps to create and start an Angel process.

- Copy the sample Angel JCL from member BAQZANGL in SBAQSAMP to your PROCLIB.
- Rename the procedure so it matches the STARTED profile you created for the Angel.
- Customize the Angel JCL:

```
//BBGZANGL PROC PARMS='',COLD=N,NAME='' 2
//*-------------------------------------------------------------------
//  SET ROOT='/u/MSTONE1/wlp' 1
//*-------------------------------------------------------------------
//* Start the Liberty angel process
//*-------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*-------------------------------------------------------------------

//STEP1   EXEC PGM=BPXBATA2,REGION=0M,TIME=NOLIMITE,
//  PARM='PGM &ROOT./lib/native/zos/s390x/bbgzangl COLD=&COLD NAME=X
//             &NAME &PARMS'
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
//* ================================================================= */
```

Notes:

1. Change the *SET ROOT=* value so it reflects the install location for z/OS Connect EE, *including* the */wlp* sub-directory.
2. A name can be given to an Angel either by providing a value in the NAME parameter in the JCL by overriding the N*AME* parameter when the Angel is started, e.g. ***S BAQZANGL,NAME=PRODUCTION***

- Start the Angel with MVS command ***S angelProc***
- Verify the Angel received the authorization ID you intended. This validates the STARTED profile you created for the Angel process.

  Then start the server and verify basic operations:

- Start the z/OS Connect server with the following command

  S serverProc,PARMS='serverName'

9 If you see "CWWKB0307E: The angel process on this system is not compatible with the local communication service. The current angel version is 2, but the required angel version is 3," then update the existing Angel JCL start proc to point to z/OS Connect EE and restart the Angel.

Notes: The PARMS= value is case sensitive. Issue this command in the z/OS "command extensions" (a single slash in SDSF) to preserve the case. Otherwise entering ***/S proc,PARMS='servername'*** in SDSF will fold the entire command to uppercase including the *servername*. If you want to simplify the start command ***/S proc***, then update the first line of the JCL procedure and include the server name in the PARM= parameter on the first line. Then when you issue /S proc the *PARMS='servername'* will be derived from the first line of the JCL.

Where *serverProc* is the name you gave your z/OS Connect EE server JCL start procedure, and *serverName* is the name you gave your created server.

☐ Verify the server received the authorization ID you intended. This validates the STARTED profile you created for the server.

☐ Go to the */var/zosconnect/servers/serverName/logs* directory

☐ Look in the messages.log file. You should see the following messages[10]. See notes that follow:

```
CWWKE0001I: The server serverMame has been launched.
CWWKB0103I: Authorized service group KERNEL is available.
CWWKB0103I: Authorized service group LOCALCOM is available.
CWWKB0103I: Authorized service group SAFCRED is available.
CWWKB0103I: Authorized service group TXRRS is available.    1
CWWKB0103I: Authorized service group WOLA is available.
CWWKB0103I: Authorized service group ZOSDUMP is available.
CWWKB0103I: Authorized service group ZOSWLM is available.
CWWKB0104I: Authorized service group PRODMGR is available.
CWWKB0104I: Authorized service group ZOSAIO is available.              2
CWWKB0103I: Authorized service group CLIENT.WOLA is available.
CWWKB0108I: IBM Corp product z/OS Connect version 03.00 successfully registered
with z/OS.
CWWKB0113I: The number of successfully registered products with z/OS is 1. These
products will deregister from z/OS when the address space terminates.
….
CWWKO0219I: TCP Channel defaultHttpEndpoint has been started and is now listening
for requests on host *  (IPv6) port port.   3
CWWKS4105I: LTPA configuration is ready after 0.612 seconds.
BAQR0000I: z/OS Connect Enterprise Edition version 3.0.15.0 (20181120-1404).
CWWKF0012I: The server installed the following features: [servlet-3.1, ssl-1.0,
jndi-1.0, json-1.0,zosconnect:zosConnect-2.0, distributedMap-1.0, appSecurity-2.0,
jaxrsClient-2.0].          4
CWWKF0008I: Feature update completed in 3.101 seconds.
CWWKF0011I: The server server_name ready to run a smarter planet.
SRVE0169I: Loading Web Module: z/OS Connect.    5
BAQR0000I: z/OS Connect Enterprise Edition version 3.0.0.1 (20170621-0908)
SRVE0250I: Web Module z/OS Connect has been bound to default_host.
CWWKT0016I: Web application available (default_host): http://<host>:<port>/ 6
```

Notes:

1. The "Authorized service group" messages indicate the success of the server to access the Angel process with the SERVER profiles you created.
2. Some *Authorized service group* messages may not be available depending on what SERVER profiles you created and whether the server ID was granted READ to the profile.
3. You should see your HTTP port show up in this message.

10 The messages may occur in a slightly different order. That's okay; the important thing is the various success indicators are present.

4. You should see *zosconnect:zosConnect-2.0* show up in the features that were installed.
5. The z/OS Connect web module should show loaded

If your server looks good at this point, then proceed by adding basic security.

Setup of basic security

Key Point: We will keep this as simple as possible at this phase of setup and validation. We do that because we want to get you to the definition of services and APIs as quickly and easily as possible. The security setup we illustrate here works but is definitely not suitable for anything but testing purposes. If you are interested in seeing how to enable SAF to perform these security functions, see Go to *Db2 REST services security*.

This section covers a few topics related to Db2 REST services security.

- SAF class DSNR

  Access to Db2 REST services requires READ access to the Db2 subsystem DSNR resource. If a request for Db2 REST services fails to Db2 subsystem DSN2 with this message:

  Simply permit READ access to this resource to the identity in question, e.g.

```
PERMIT DSN2.REST CLASS(DSNR) ID(USER2) ACC(READ)
SETROPTS RACLIST(DSNR) REFRESH
```

- Db2 package access

  If a user is not able to display a valid Db2 REST services in the z/OS Connect Db2 services development tooling or by using a POST to the Db2 provided REST interface URL of http://wg31.washington.ibm.com:2446/services/DB2ServiceDiscover, then they may not have sufficient access to the package containing the service.

  For example, if service zCEEService.selectEmployee is defined to Db2 but not visible in the z/OS Connect tooling or if a GET request to URL

  http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee fails with message:

  The user needs to be granted execute authority on package zCEEService.*selectEmployee with* command:

  GRANT **`EXECUTE ON PACKAGE "zCEEService"."selectEmployee"`** or

```
GRANT EXECUTE ON PACKAGE "zCEEService"."*"
```

Using SAF for registry and access role checking on page 86

Here you will set up security definitions in the server.xml to provide the minimum required (by default).

Do the following:

☐ *Go* to the */var/zosconnect/servers/**serverName*** directory

☐ Edit the *server.xml* file.

☐ Add an include element as show below:

```
<server description="new*server">
<include location="/var/zcee/basic.xml" optional="true"/>

  <!-- Enable features -->
    <featureManager>
```

☐ Create a file *basic.xml* in directory */var/zcee* and add the XML shown here, (see notes that follow)

```
<server description="basic security">

    <!-- Enable features -->
    <featureManager>
        <feature>appSecurity-2.0</feature>   1
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>  2

    <webAppSecurity allowFailOverToBasicAuth="true" />    3

    <basicRegistry id="basic1" realm="zosConnect">       4
        <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="zos.connect.access.roles">   5
        <security-role name="zosConnectAccess">
            <user name="Fred"/>
        </security-role>
    </authorization-roles>

</server>
```

1. Enables application security, which z/OS Connect EE uses[11].

2. Enables use of a default key/trust store generated by Liberty. This allows SSL from the REST client to z/OS Connect EE without having to introduce the complexity of creating and managing certificates at this point.

3. This will result in a userid and password prompt at the REST client, rather than using the default client certificate mechanism.

4. This defines a user registry with a single entry of Fred and a password.

5. IBM z/OS Connect EE requires the authenticated user to have role access as well. This provides that access.

☐ Save the files.

11 This is redundant. If you look at the `messages.log` output from earlier, you will see that `appSecurity-2.0` is loaded automatically. That's because z/OS Connect EE was loaded, and application indicated it needed `appSecurity-2.0`. So, Liberty auto-loaded it. Including it as a `<feature>` does not hurt. It is a good visual reminder of key features required by z/OS Connect EE 2.

☐ Enter a MVS modify command to refresh the configuration, e.g. ***F serverProc,ZCON,REFRESH***

The messages below should appear in the messages.log file

```
CWWKG0016I: Starting server configuration update.
CWWKG0028A: Processing included configuration resource: /zcee/basic.xml
CWWKF0008I: Feature update completed in 1.134 seconds.
CWPKI0803A: SSL certificate created in 2.203 seconds. SSL key file: /var/zosconnect/servers/zceetest/resources/security/key.jks
CWWKS9112A: The web application security settings have changed. The following properties were modified: allowFailOverToBasicAuth=true
CWWKG0017I: The server configuration was successfully updated in 2.380 seconds.
```

☐ Use a web browser and enter the following URL:

https://*<host>*:*<port>*/zosConnect/apis

where:

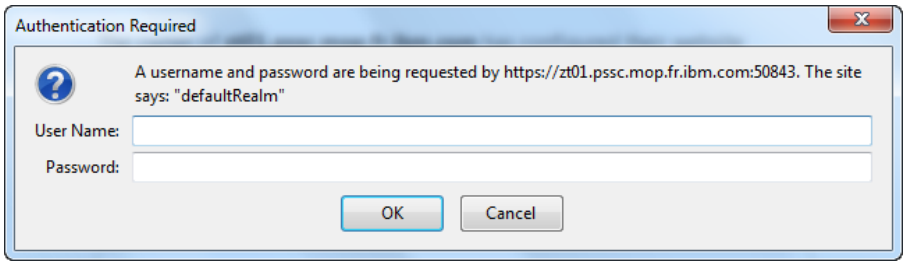
- The protocol is http**s** (note the "s")
- *<host>* is the TCP host for your server
- *<port>* is the secure port (httpsPort=) for your server
- The "C" in "zosConnect" is in uppercase (otherwise you'll get a 404 not found error)

☐ Your browser will challenge the security of the connection because the certificate authority that signed the server certificate is the default Liberty CA, and your browser does not recognize that. Accept the challenge[12].

Tech Tip: Accessing the server using a browser can be done at this stage. But be aware the server is using a self-signed certificate at this time and some browsers will not always accept self-signed certificates. If this is an issue download and install the cURL tool ,see section *Testing z/OS Connect Services Using cURL* on page *74* and follow the instructions later in this section for using cURL to verify the server.

12 This will create an error in `messages.log` and an FFDC directory with entries there to capture the error. This is expected.

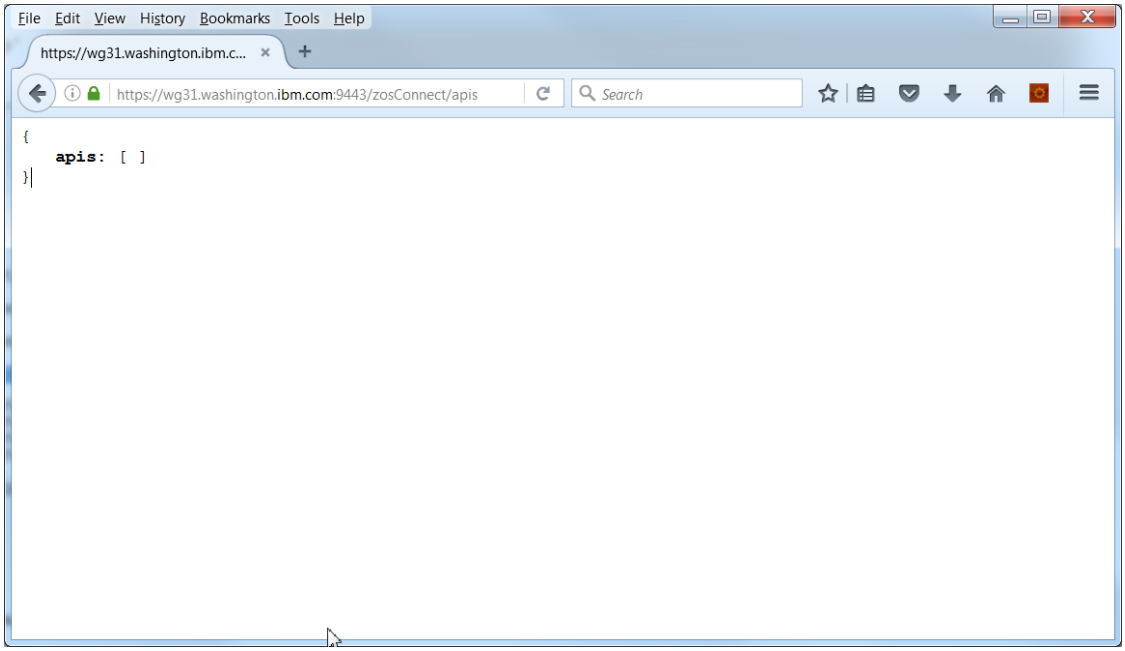
☐ You should then get a basic authentication prompt:



This is because of the *allowFailOverToBasicAuth="true"* in the server.xml.

Provide the userid and password you supplied for the basicRegistry entry in the server.xml file: **Fred** and **fredpwd** (*this is case sensitive*).

☐ You should then see a screen like the following:



Tech Tip: The browser add-on or plug-in *JSONView* has been installed in this browser. This add-on formats JSON messages so they are easier to read and enables hyperlinks, etc. The browser screen shots in this document show the effects of this browser add-on.

That is telling you z/OS Connect sees no APIs are currently configured. That is a good sign at this point – it is telling you the Liberty z/OS server recognizes that z/OS Connect EE V3.0 is in fact active, but no APIs are currently present.

☐ Stop the server with **/P** ***<server_proc>*** [13]. This will give you a clean messages.log on the next start, which makes it easier to look for and find the key success messages.

13 It's really `/P <jobname>`, but earlier you started the server with just the proc name, so that becomes the jobname as well.

The essentials are in place for you to begin coding up services and using the API editor to create the API artifacts.

☐ Optionally, verify the server using cURL. For details regarding this test tool see *Testing z/OS Connect Services Using cURL* on page 74.

curl -X GET --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://<host>:<port>/zosConnect/apis

Use following URL:

https://*<host>*:*<port>*/zosConnect/apis

where:

- The protocol is http**s** (note the "s")
- *<host>* is the TCP host for your server
- *<port>* is the secure port (httpsPort=) for your server
- *Note:* The "C" in "zosConnect" is in uppercase (otherwise you'll get a 404 not found error

☐ You should then see something like the following:

> ***curl -X GET --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://wq31.washington.ibm.com:9443/zosConnect/apis***
>
> {"apis":[]}

Tech Tip: If the above test fails adding a -v flag to the curl command will provide a trace that may be useful in resolving the cause of the failure.

This is telling you z/OS Connect sees no APIs are currently configured. That is a good sign at this point – it is telling you the Liberty z/OS server recognizes that z/OS Connect EE is in fact active, but no APIs are currently present.

☐ Stop the server with MVS command **P *serverProc*** [14]. This will give you a clean messages.log on the next start, which makes it easier to look for and find the key success messages.

The essentials are in place for you to begin coding up services and using the API editor to create the API artifacts.

14 It's really `/P <jobname>`, but earlier you started the server with just the proc name, so that becomes the jobname as well.

Installing the z/OS Connect EE V3.0 tooling

That tooling is called *z/OS Connect EE API Toolkit,"* and it is an Eclipse-based tool for creating services and editing API definitions.

There are two steps to this process: (1) installing an Eclipse platform, and (2) installing the z/OS Connect EE API Toolkit into the Eclipse platform[15]. We will go into detail on how to install into an instance of IBM Explorer for z/OS.

Installing an Eclipse runtime platform

The z/OS Connect EE API Editor is a plugin tool to an Eclipse platform, such as:

- One of the eclipse.org packages (e.g. Neon or later)
- IBM Explorer for z/OS Aqua 3.1

  If you already have one of these installed, then you may jump to the next section.

The following are the instructions for installing IBM Explorer for z/OS Aqua 3.1

☐ Go to the following URL: https://developer.ibm.com/mainframe/products/downloads/

☐ Follow the instructions you find there to install using either IBM Installation Manager or "from scratch" (meaning you do not have Installation Manager).

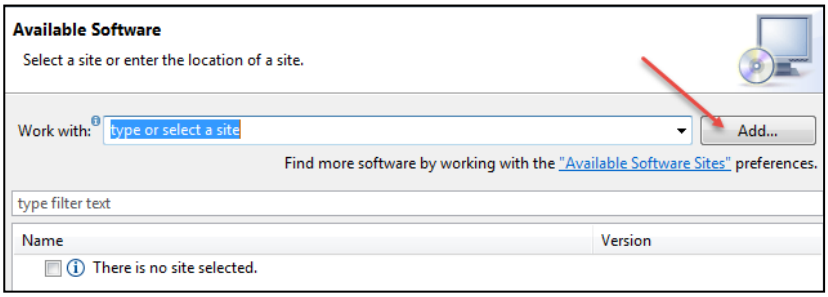
☐ When completed, start IBM Explorer for z/OS Aqua 3.1.

Installing the z/OS Connect EE V3.0 API Toolkit

The IBM z/OS Connect EE API Toolkit is a plugin that is installed into an Eclipse environment. Installing the plugin is a relatively simple thing:

☐ From your open Eclipse platform[16], select *Help →Install New Software*.

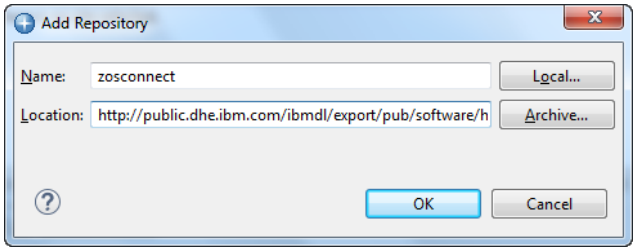
☐ Then click the "Add" button:



15 KC: https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/com.ibm.zosconnect.doc/installing/install_explorer.html
16 Either IBM z/OS Explorer or from an Eclipse installation.

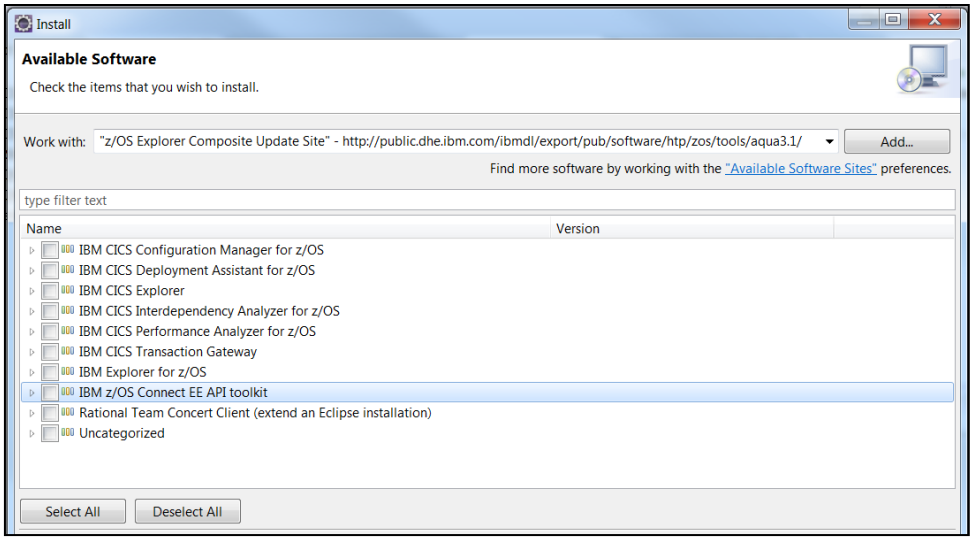
 Provide a name (such as ***"z/OS Explorer Composite Update Site"*** and then for *Location* provide this URL:

http://public.dhe.ibm.com/ibmdl/export/pub/software/htp/zos/tools/aqua3.1/

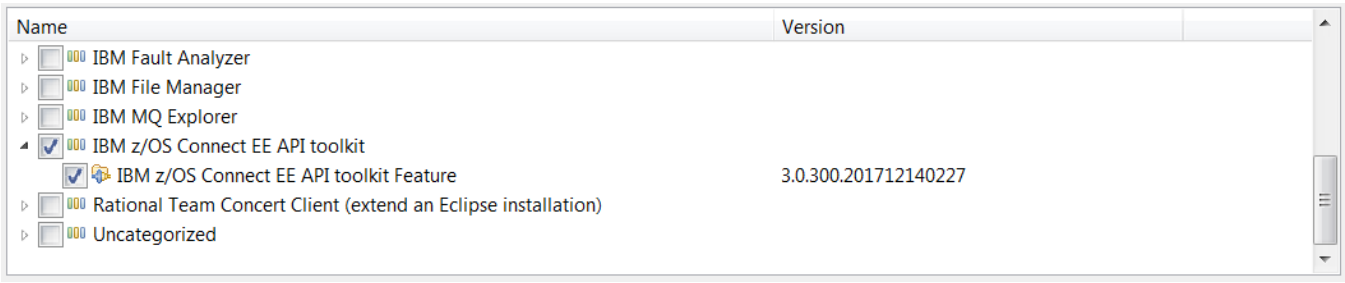


 Click **OK**.

 It will spend a little time searching for the tools available at that location. You will see a *Pending* indicator. Then it will populate the window with something like this:

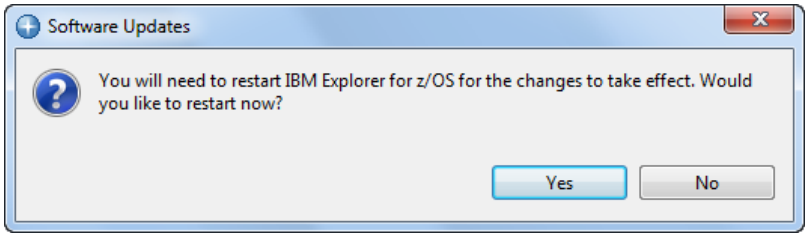


 Scroll down, locate and check the box beside *IBM z/OS Connect EE API Toolkit*:

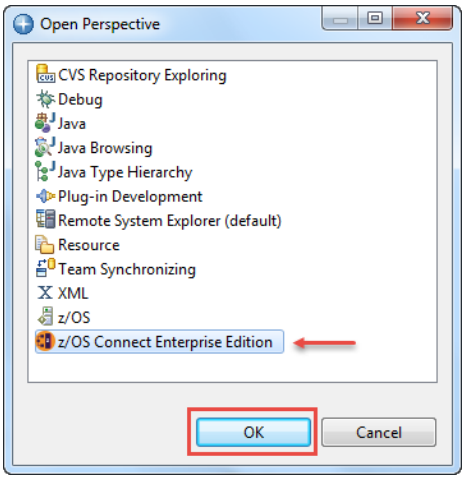


 Click **Nex**t twice and then agree to the license agreement. Then click **Finish**.

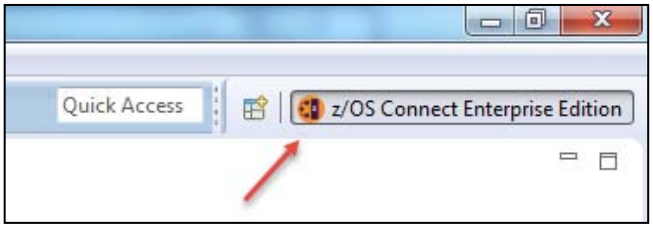
 When the installation is complete, there will be a message that you need to restart, click **Yes** to continue.



 Eclipse will restart. When it is open, select *Window* → *Open Perspective* → Other and select *z/OS Connect Enterprise Edition*. Click **OK** to continue.

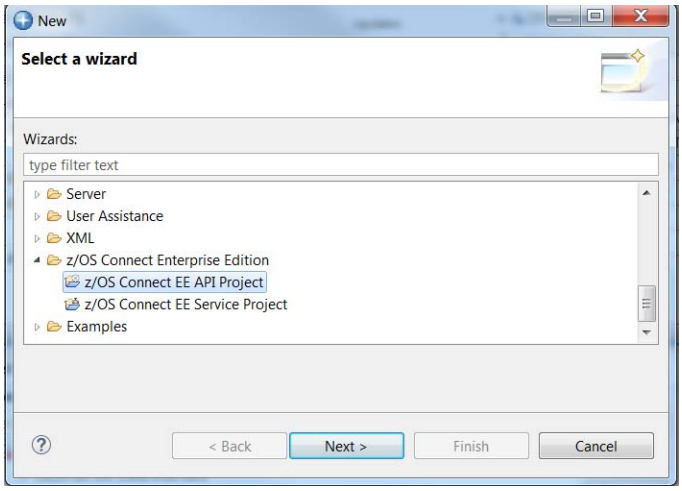


 In the upper-right corner you should see something like this:



Note: there may other *perspectives* showing there. The key is seeing the *z/OS Connect Enterprise Edition* perspective indicated and highlighted.

 Now click *File* → *New* →*Other*, then scroll down, open the folder *z/OS Connect Enterprise Edition* and *look for z/OS Connect EE API Project* and *z/OS Connect EE Service Project* as shown below:



Note: this verifies that the plugin is installed and ready to use. You will use the API Toolkit later in the install/setup process.

 Click *Cancel*. Close Eclipse if you wish.

Checkpoint: status at this point

At this point you have:

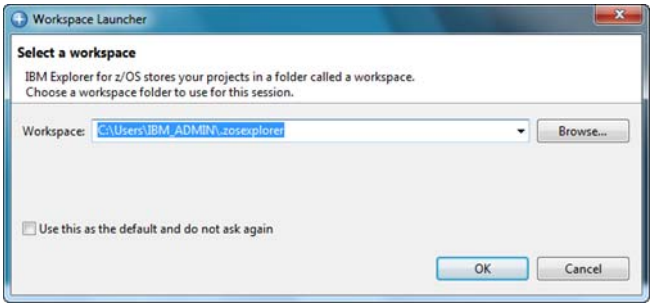
- z/OS Connect EE V3 installed
- z/OS Connect EE V3 Toolkit installed
- Key SAF profiles created
- A server created and capable of starting as a z/OS started task
- The basic security structure is in place for z/OS Connect EE

Open IBM z/OS Explorer for z/OS and connect to the z/OS Connect EE server

N.B. In these sections there will be references in text and screen shots to real host names and ports, directory structures specific to the system used to develop this material. These are only provided in the context of working samples.

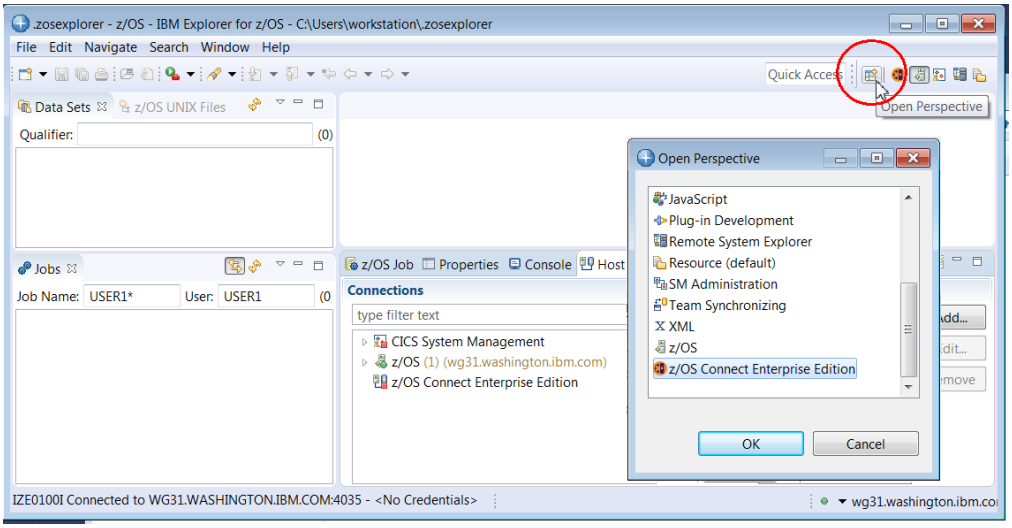
 On the workstation desktop, locate the *IBM Explorer for z/OS* icon and double click on it to open the tool.

 You will be prompted for a workspace:

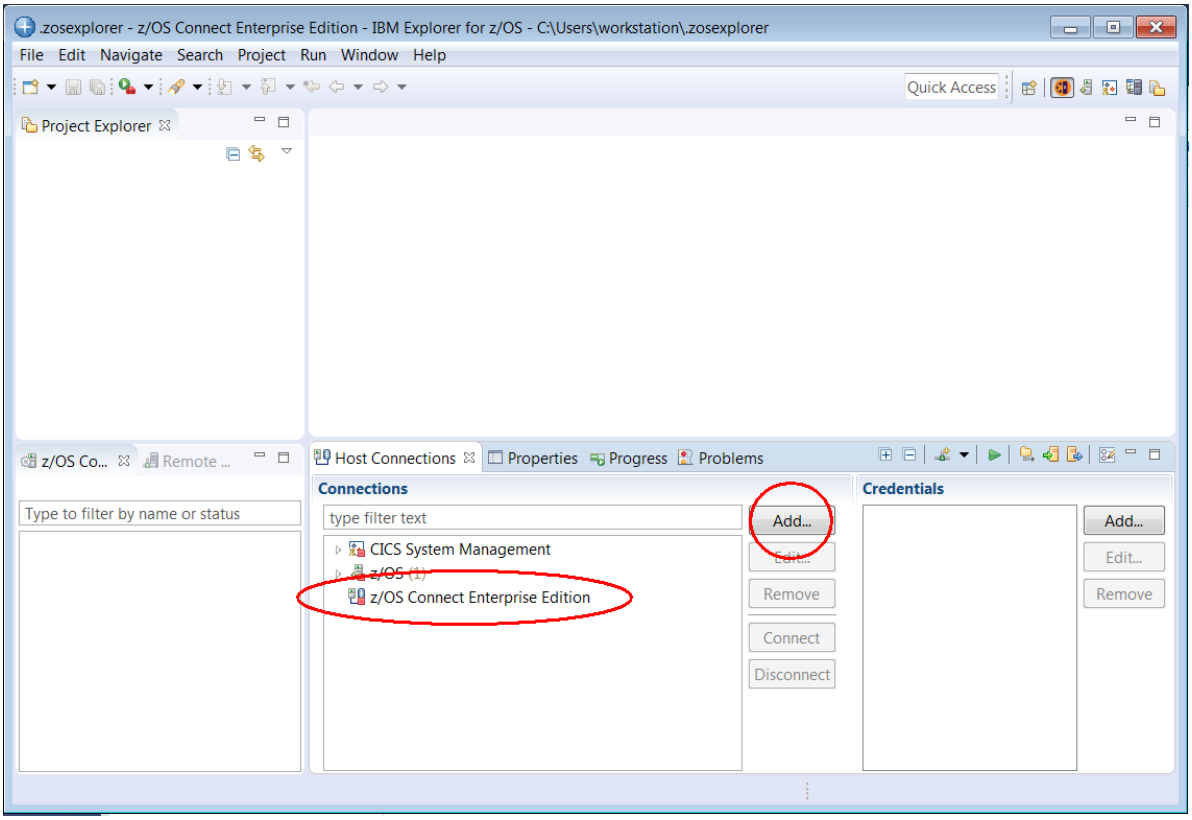


Take whatever default value is seen and click **OK**. If you see a *Welcome* tab close it by click on the white X in the tab.

 If the current perspective is not *z/OS Connect Enterprise Edition*, select the *Open Perspective* icon on the top right side to display the list of available perspectives, see below. Select **z/OS Connect Enterprise Edition** and click the **OK** button to switch to this perspective.



 To add a connection to the z/OS Connect Server select *z/OS Connect Enterprise Edition* connection in the *Host connection*s tab in the lower view and then click the **Add** button.



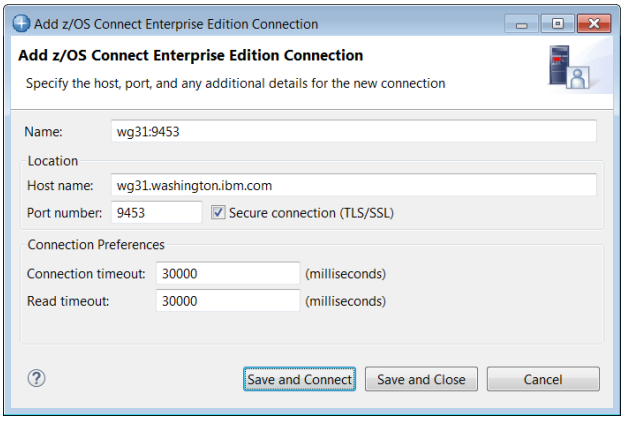
Tech-Tip: Eclipse base development tools like IBM z/OS Explorer; provide a graphical interface consisting of multiple views within a single window.

A view is an area in the window dedicated to providing a specific tool or function. For example, in the window above, *Host Connections* and *Project Explorer* are views that use different areas of the window for displaying information. At bottom on the right there is a single area for displaying the contents of four views stacked together (commonly called a *stacked views*), *z/OS Host Connections*, *Properties*, *Progress* and *Problems*. In a stacked view, the contents of each view can be displayed by clicking on the view tab (the name of the view).

At any time, a specific view can be enlarged to fill the entire window by double clicking in the view's title bar. Double clicking in the view's title bar will be restored the original arrangement. If a z/OS Explorer view is closed or otherwise disappears, the original arrangement can be restored by selecting Windows → Reset Perspective in the window's tool bar.

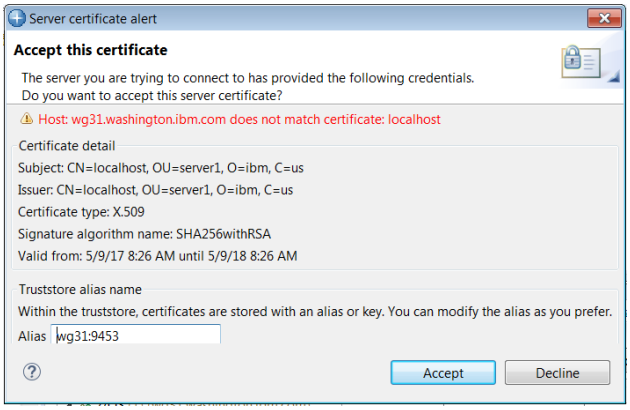
Eclipse based tools also can display multiple views based on the current role of the user. In this context, a window is known as a perspective. The contents (or views) of a perspective are based on the role the user, i.e., developer or administrator.

 In the pop-up list displayed select *z/OS Connect Enterprise Edition* and on the *Add z/OS Connect Enterprise Edition Connection* screen enter ***wg31.washington.ibm.com*** for the *Host name*, *9453* for the *Port Number*, check the box for *Secure connection (TLS/SSL)* and then click the **Save and Connect** button.



 On the *z/OS Connect Enterprise Edition – User ID* required screen create new credentials for a *User ID* of ***Fred*** and a *Password or Passphrase* of ***fredpwd*** (case matters). Remember the server is configured to use basic security. If SAF security had been enabled, then a valid RACF User ID and password will have to be used instead. Click **OK** to continue.

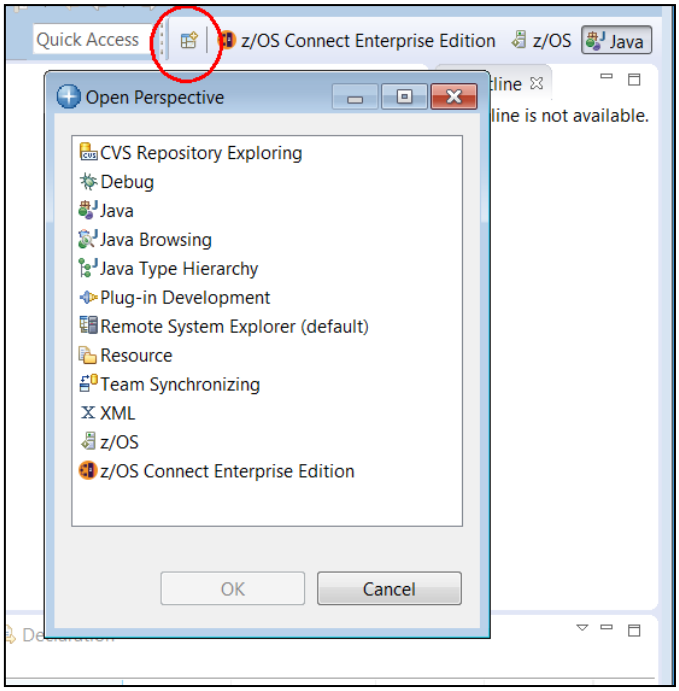
 Click the **Accept** button on the *Server certificate alert – Accept this certificate* screen. You may be presented with another prompt for a userid and password, enter ***Fred*** and ***fredpwd*** again.



 The status icon beside *wg31:9453* should now be a green circle with a lock. This shows that a secure connection has been established between the z/OS Explorer and the z/OS Connect server. A red box indicates that no connection exists.

A connection to the remote z/OS system was previously added. In the *Host Connection* view expand *z/OS Remote System* under *z/OS* and select *wg31.washington.ibm.com*. If the connection is not active the **Connect** button will be enabled. Click the **Connect** button and this will establish a session to the z/OS system. This step is required when submitting job for execution and viewing the output of these jobs later in this exercise

 First establish a connection to your z/OS Connect server. Select the *Open Perspective* icon on the top right side to display the list of available perspectives. Select z/OS and click the **OK** button.



CICS RESTful APIs

Connectivity between the z/OS Connect EE (zCEE) server and a CICS region is provided by CICS *IP Interconnectivity* (IPIC). Further CICS configuration may be required.

In the sample application that will be shown, the CICS region is running on TCP/IP host *wg31.washington.ibm.com* and has an IPIC TCPIPService listening on port *1491*. The z/OS Connect EE server is running on the same TCP/IP host and is listening on port *9443* for HTTPS requests.

Adding IPIC support to a z/OS Connect server

Do the following:

□ Go to the server.xml directory, e.g. */var/zosconnect/servers/serverName*

□ Edit *server.xml* and add the lines highlighted here in **bold** as shown, see the notes below:

```
 Enable features -->
   <featureManager>
      <feature>zosconnect:zosConnect-2.0</feature>
      <feature>zosconnect:zosConnectCommands-1.0</feature>
      <feature>zosconnect:cicsService-1.0</feature>
</featureManager

<zosconnect_cicsIpicConnection id="catalog"     1
     host="wg31.washington.ibm.com"        2
     port="1491"/>                 3
```

Notes:

1. This value must match the value that is specified for the *connectionRef* property when a *service* is developed in the API Toolkit.
2. The TCP/IP host name or IP address of the host on which the CICS region is running.
3. The port assigned to the IPIC TCPIPService defined in the CICS region.

□ Save the file.

Install the Catalog Manager Sample in the CICS region

For this document we are using the CICS "catalog manager" sample application. This application simulates an office supplies store application. It is useful for illustrating z/OS Connect EE because it is plausibly "real world" while not being overly-complex.

The details of this CICS sample application are provided here:

https://www.ibm.com/support/knowledgecenter/SSGMCP_5.4.0/applications/example-application/dfhxa_t100.html

Work with the CICS administrator and do the following:

☐ Enable the catalog manager sample application based on the instructions provided in the URL given above.

☐ Verify the sample application is functional by accessing it with the transaction ***EGUI*** from a 3270 CICS terminal session.

The steps that follow will guide you through creating the service and API definitions to access that sample application using REST and z/OS Connect EE.

Setup of IPIC support in a CICS region

Adding support for IPIC in a CICS region is quite simple. First, the CICS region must have

- TCPIP=YES and
- ISC=YES

Specified as system initialization parameters at CICS startup.

Finally, a CICS *TCPIPService* needs to be defined and installed in the CICS region. This resource identifies which port the CICS region will listen on for inbound IPIC requests.

This resource should have these attributes:

TCPIPService resource attribute	**Value required**
URM	DFHISAIP
Port Number	A numeric value of an available port, e.g. 1491
Status	OPEN
Protocol	IPIC
Transaction	CISS

Tech Tip: In this scenario we will be using the API name for the connection reference property. The rationale is that the connection reference property is information which is integral information about the API. The developer will be setting this property during development of the service and should provide this name to the administrator responsible for configuring CICS connections in the z/OS Connect EE server.
When multiple services are deployed in the same server there maybe multiple *cicsIpicConnection* connecting to the same or different CICS regions. Each tailored to the specified requirements of the API or service, e.g. and to the requirements of the infrastructure, e.g. security, number of send/receive sessions, etc. Or alternatively there could just be one connection defined and every service uses the same value for the connection reference property.
Multiple IPIC connections to the same CICS region seems to work with no issues as long as identity propagation has not been enabled between the z/OS Connect server and the CICS region. Configuring identity propagation should be done over a dedicated TCPIPService port.

Developing RESTful Services for CICS

Once the IPIC configuration is completed follow the instructions for the development and deployment of services in the ***Developing RESTful APIs for CICS COMMAREA program*** document at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop. This document shows how to develop and deploy CICS services as well as showing how to develop and deploy APIs that consume these services. For the purposes of this document we are only interested in deploying and testing services, but feel free to develop and test APIs also.

Test the Services

If you have followed the instructions in ***Developing RESTful APIs for CICS*** *you* should have at least 3 services deployed to the server. These services are *inquireSingle*, *inquireCatalog* and *placeOrder*. The services can be used to test connectivity to CICS from the z/OS Connect server. The services and infrastructure should be tested before developing an API to ensure the infrastructure and the request and response messages are as expected.

Follow the instructions for testing services in either section *Testing z/OS Connect Services Using Postman* on page 68 or section *Testing z/OS Connect Services Using cURL* on page 74 to test the 3 services.

• For service *inquireSingle* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?action=invoke* and JSON request message:

```
{
  "DFH0XCP1": {
     "inquireSingle": {
      "itemID": 20,
      }
   }
}
```

With expected JSON response message:

```
{"DFH0XCP1":{"CA_RESPONSE_MESSAGE":"RETURNED ITEM: REF
=0020","CA_INQUIRE_SINGLE”:{"CA_SINGLE_ITEM":{"CA_SNGL_ITEM_REF":20,"CA
_SNGL_DESCRIPTION":"Ball Pens Blue 24pk", "CA_SNGL_DEPARTMENT":10,
"IN_SNGL_STOCK":6, “CA_SNGL_COST":"002.90", "ON_SNGL_ORDER":50}},
```

• For service *inquireCatalog* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/inquireCatalog?action=invoke* and JSON request message.

```
{
  "DFH0XCP1": {
     "inquireCatalog": {
      "startItemID": 20
      }
   }
}
```

With expected JSON response message:


```
{"DFH0XCP1":{"CA_RESPONSE_MESSAGE":"+15 ITEMS RETURNED","CA_INQUIRE_REQUEST":{"C
A_LAST_ITEM_REF":150,"CA_CAT_ITEM":[{"ON_ORDER":0,"CA_ITEM_REF":10,"CA_COST":"00
2.90","IN_STOCK":135,"CA_DESCRIPTION":"Ball Pens Black 24pk","CA_DEPARTMENT":10}
,{"ON_ORDER":50,"CA_ITEM_REF":20,"CA_COST":"002.90","IN_STOCK":6,"CA_DESCRIPTION
":"Ball Pens Blue 24pk","CA_DEPARTMENT":10},{"ON_ORDER":0,"CA_ITEM_REF":30,"CA_C
OST":"002.90","IN_STOCK":106,"CA_DESCRIPTION":"Ball Pens Red 24pk","CA_DEPARTMEN
T":10},{"ON_ORDER":0,"CA_ITEM_REF":40,"CA_COST":"002.90","IN_STOCK":80,"CA_DESCR
IPTION":"Ball Pens Green 24pk","CA_DEPARTMENT":10},{"ON_ORDER":0,"CA_ITEM_REF":5
0,"CA_COST":"001.78","IN_STOCK":83,"CA_DESCRIPTION":"Pencil with eraser 12pk","C
A_DEPARTMENT":10},{"ON_ORDER":40,"CA_ITEM_REF":60,"CA_COST":"003.89","IN_STOCK":
13,"CA_DESCRIPTION":"Highlighters Assorted 5pk","CA_DEPARTMENT":10},{"ON_ORDER":
20,"CA_ITEM_REF":70,"CA_COST":"007.44","IN_STOCK":101,"CA_DESCRIPTION":"Laser Pa
per 28-lb 108 Bright 500\/ream","CA_DEPARTMENT":10},{"ON_ORDER":0,"CA_ITEM_REF":
80,"CA_COST":"033.54","IN_STOCK":25,"CA_DESCRIPTION":"Laser Paper 28-lb 108 Brig
```


• For service *placeOrder* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/placeOrder?action=invoke* and JSON request message:

```
{
  "DFH0XCP1": {
     "orderRequest": {
      "itemID": 70,
      "orderQuantity": 1
      }
   }
}
```

With expected JSON response message:


```
{"DFH0XCP1":{"CA_RESPONSE_MESSAGE":"ORDER SUCCESSFULLY PLACED","CA_RETURN_CODE":0}}
```


If these tests complete as expected, then the server can communicate with CICS and the infrastructure is ready for the deployment of APIs. The development, deployment and testing of APIs can proceed.

IMS RESTful APIs

If your primary interest is IMS, you *may* have jumped directly to this section to perform installation and setup. You may have to go back and perform certain setup steps from earlier. We will offer specific instructions here which sections to visit and perform.

Accessing an IMS transaction from a z/OS Connect EE (zCEE) server is done using OTMA through IMS Connect. In the example that will be shown in the section the IMS Connect task is running on TCP/IP host *wg31.washington.ibm.com* and listening on port *4000.* The z/OS Connect EE server is running on the same TCP/IP host and is listening on port *9443* for HTTPS requests.

Adding IMS Connect support to a z/OS Connect server.

Adding support IMS Connect for communications between a zCEE server and an instance of IMS Connect requires the addition of IMS mobile feature to the feature manager list of the server and the creation of additional directories and files in the server's configuration directory structure. Note that during startup of the zCEE server these IMS configuration directories and file will be automatically created if they do not already exist.

In *Server Creation* section on page 20 there was reference to an IMS mobile server creation template. You could use this template to create a zCEE server with the proper configuration for accessing IMS Connect or you could simply add feature *imsmobile:imsmobile-2.0 to* an existing zCEE server. In either case starting or restarting the server with this feature specified will cause creation of the IMS configuration directories and files. The server xml configuration will be updated with additional *include* statements (see below) will be inserted in to the server.xml. These include files reference xml files will need to be configured with the details for accessing IMS control regions and IMS transactions.

```
<include location="/var/zosconnect/servers/zceeims/resources/imsmobile-
config/interactions/ims-interactions.xml" optional="true"/>
<include location="/var/zosconnect/servers/zceeims/resources/imsmobile-
config/connections/ims-connections.xml" optional="true"/>
<include location="/var/zosconnect/servers/zceeims/resources/imsmobile-
config/services/ims-services.xml" optional="true"/>
<include location="/var/zosconnect/servers/zceeims/ims-admin-services.xml"
optional="true"/>
```

Note the *include* lines are split over two lines for display purposes. The attributes on an *include* element will normally be on one line.

☐ Look in the messages.log file for the server. You should see something like the following message indicating successful processing of the changes:

GMOIG7777I: IMS service provider (20181120-1404) for z/OS Connect Enterprise Edition initialized successfully.

Install the IMS Phone Sample in the IMS control region

For this document we are using the IMS "phonebook" sample application. This application simulates an phonebook application. It is useful for illustrating z/OS Connect EE because it is plausibly "real world" while not being overly-complex.

The details of this IMS phonebook sample application are provided here:

https://www.ibm.com/support/knowledgecenter/en/SSEPH2_15.1.0/com.ibm.ims15.doc.ins/ims_ivpsamples.htm

Work with the IMS administrator and do the following:

☐ Enable the phonebook sample application based on the information provided in the URL given above.

☐ Verify the sample application is functional by accessing it with the transaction ***/FOR IVTNO*** from a 3270-terminal session.

Verify the IMS Service Provider

The first test will use the provided *IMSPingService* service to verify z/OS Connect recognizes the service, and it recognizes the other elements of the IMS implementation.

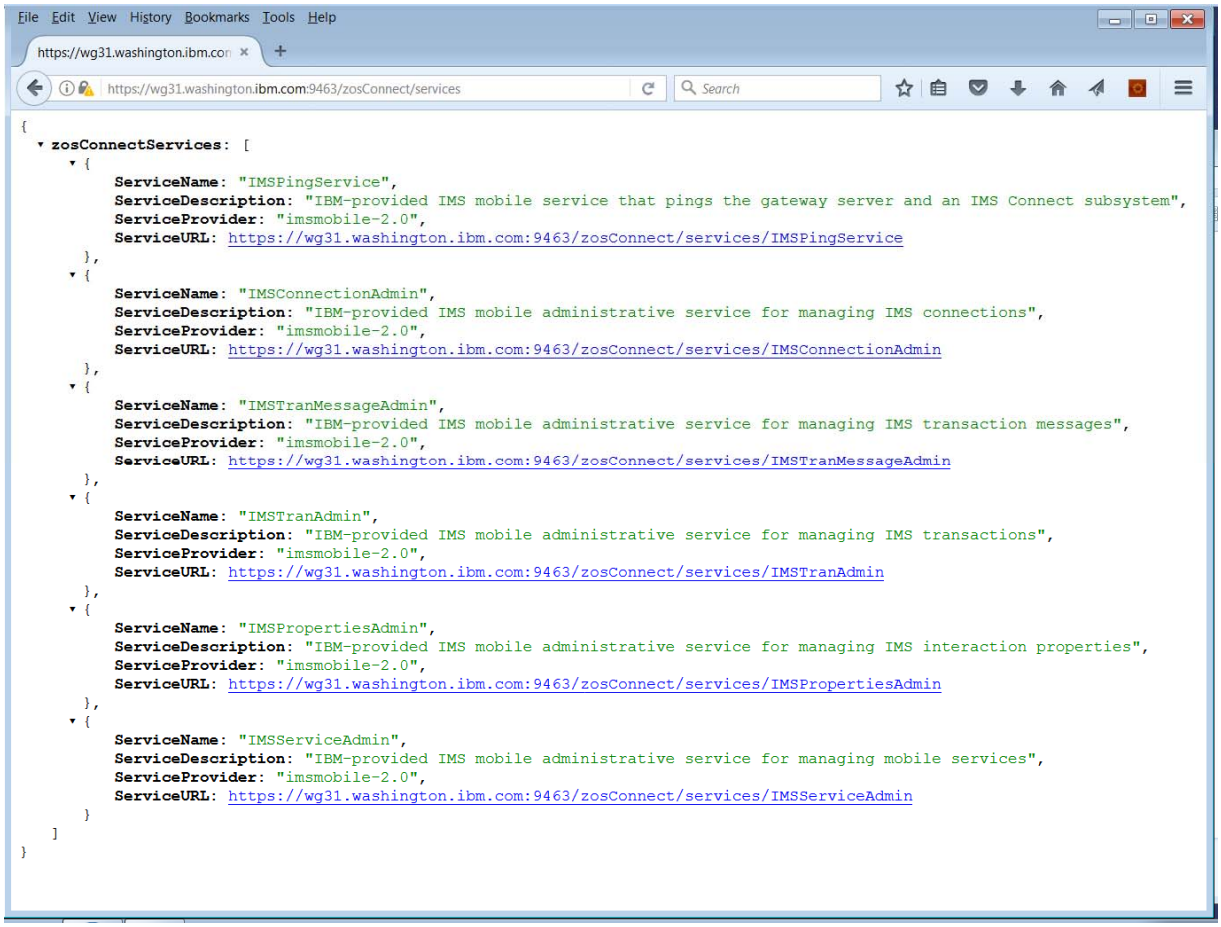
- Open a normal browser and enter the following URL:

https://wg31.washington.ibm.com:9443/zosConnect/services

> **Note:** This is done because most REST clients are not very good at handling encryption when the server certificate is self-signed, as is the case with your Liberty z/OS server now.

You should receive a certificate challenge because the server certificate is signed by a CA that is not known to the browser. Accept the challenge.

• You will then receive the basic authentication prompt. Supply the ID (***Fred)*** and password (***fredpwd***). You should receive in return a JSON string[17] that represents all the services that are auto-created with the IMS support:



Note: This test does not exercise a connection to IMS Connect. You will do that after you have configured a service and interaction definition.

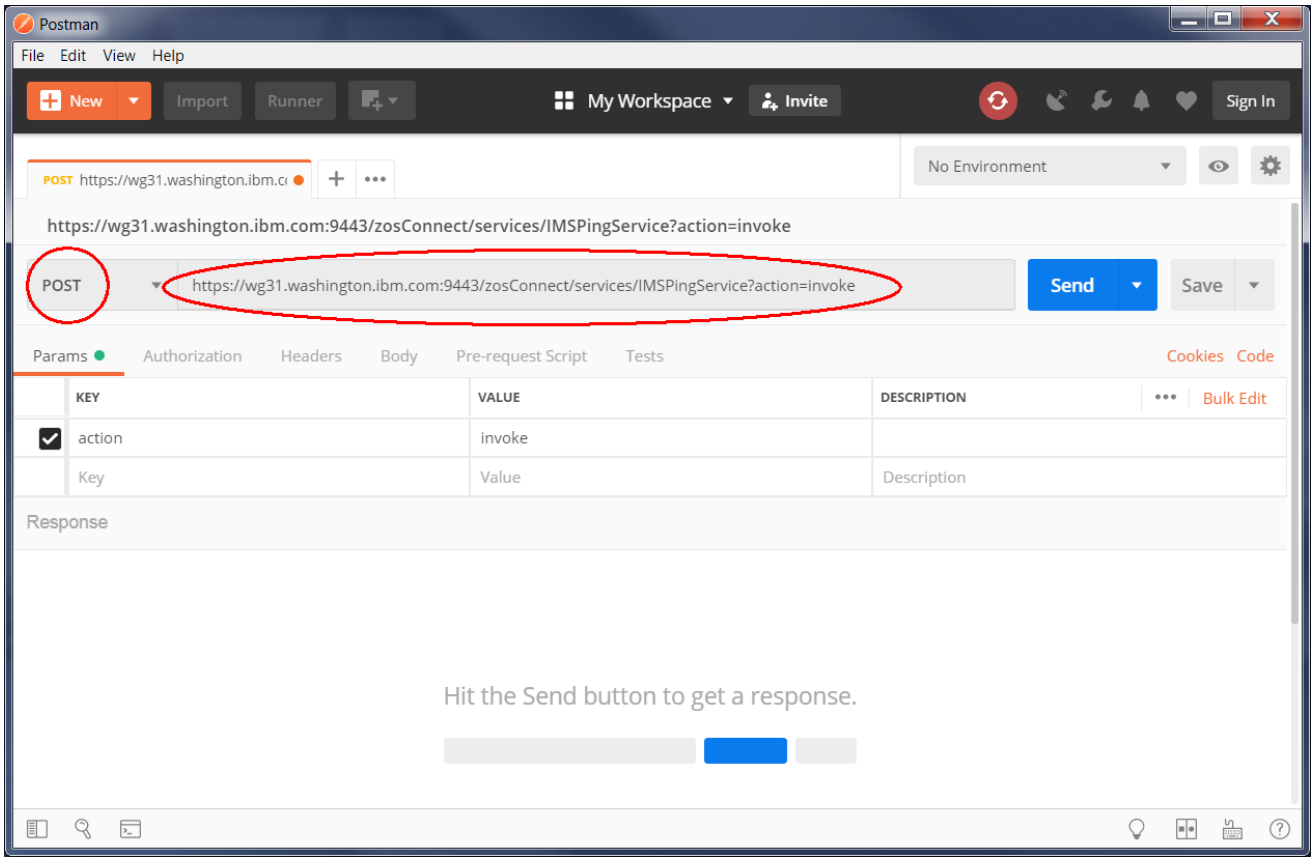
17 The browser doesn't understand how to format the JSON, a plug-in has been installed in the browser used to capture these screen shots to make the JSON easier to read.

Two products which seem to be most popular tools for testing RESTful APIs used to test the services. The two products are *Postman* which is available for downloading from https://www.getpostman.com/apps and *cURL (client URL)* which is available for downloading from https://curl.haxx.se/download.html. The use of both will be shown in this section of the exercise.

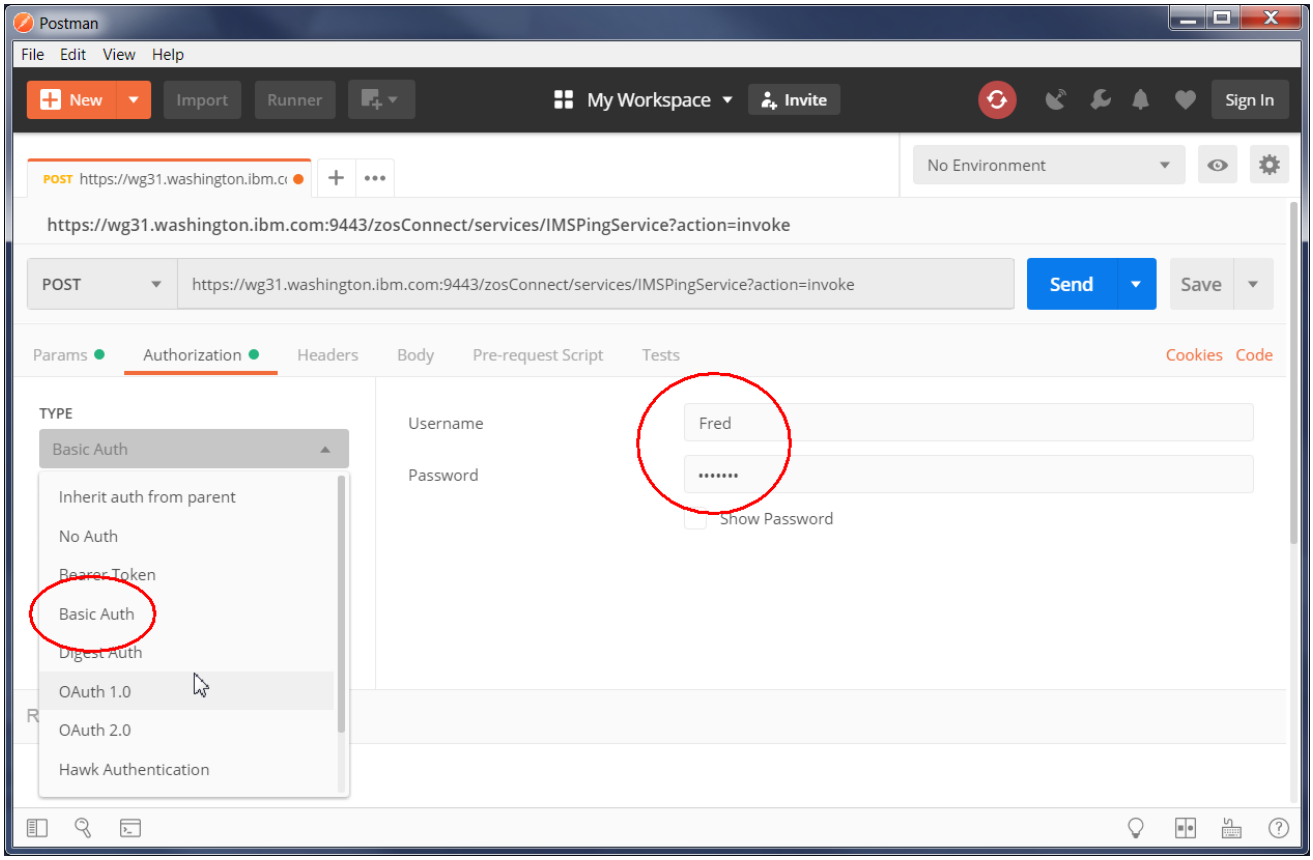
Using Postman

- To test the inquireSingle service open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages, use the down arrow to select **POST** and enter in the URL area (see below)

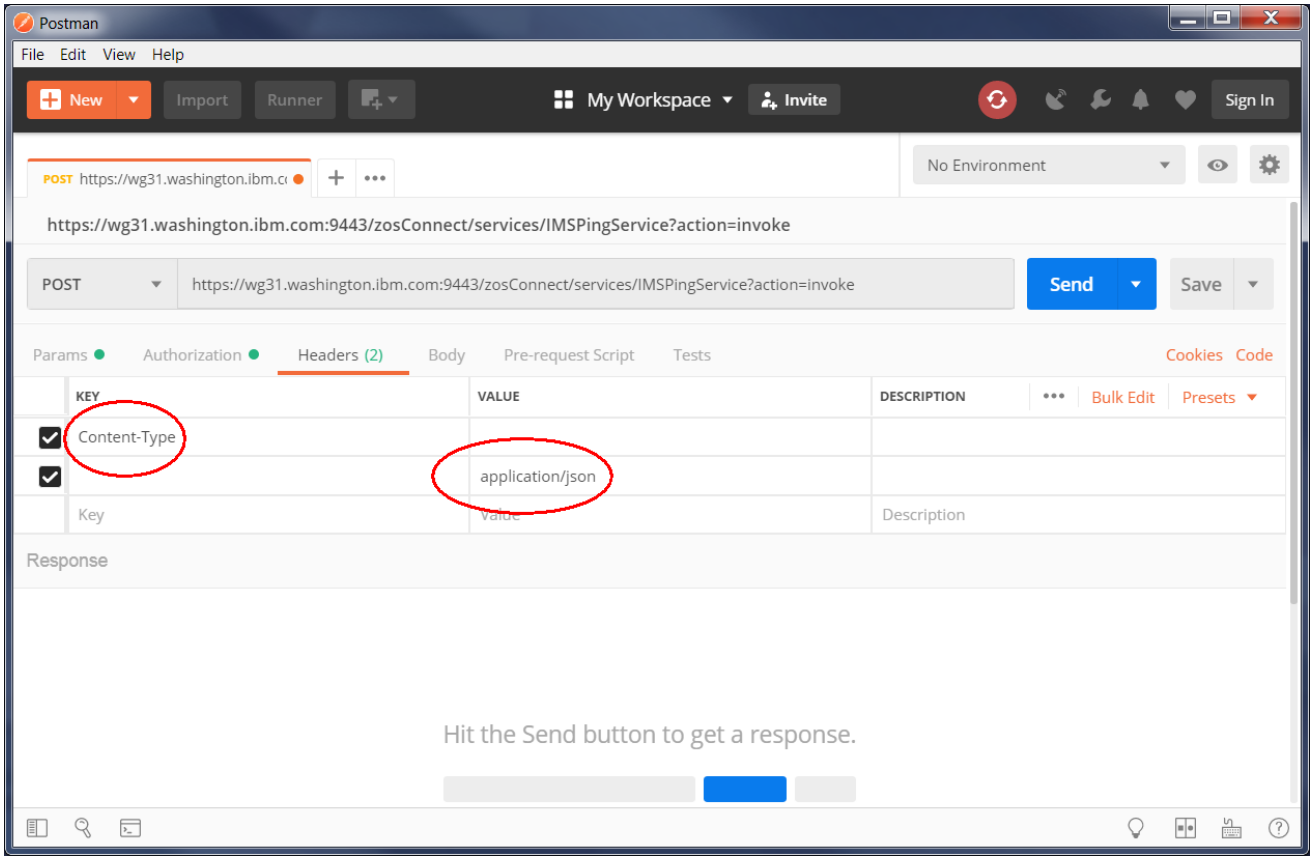
https://wg31.washington.ibm.com:9443/zosConnect/services/IMSPingService?action=invoke



• No *query* or *path* parameters are required so next select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter ***Fred*** as the username and ***fredpwd*** as the Password (these are the identity and password defined in the server.xml).

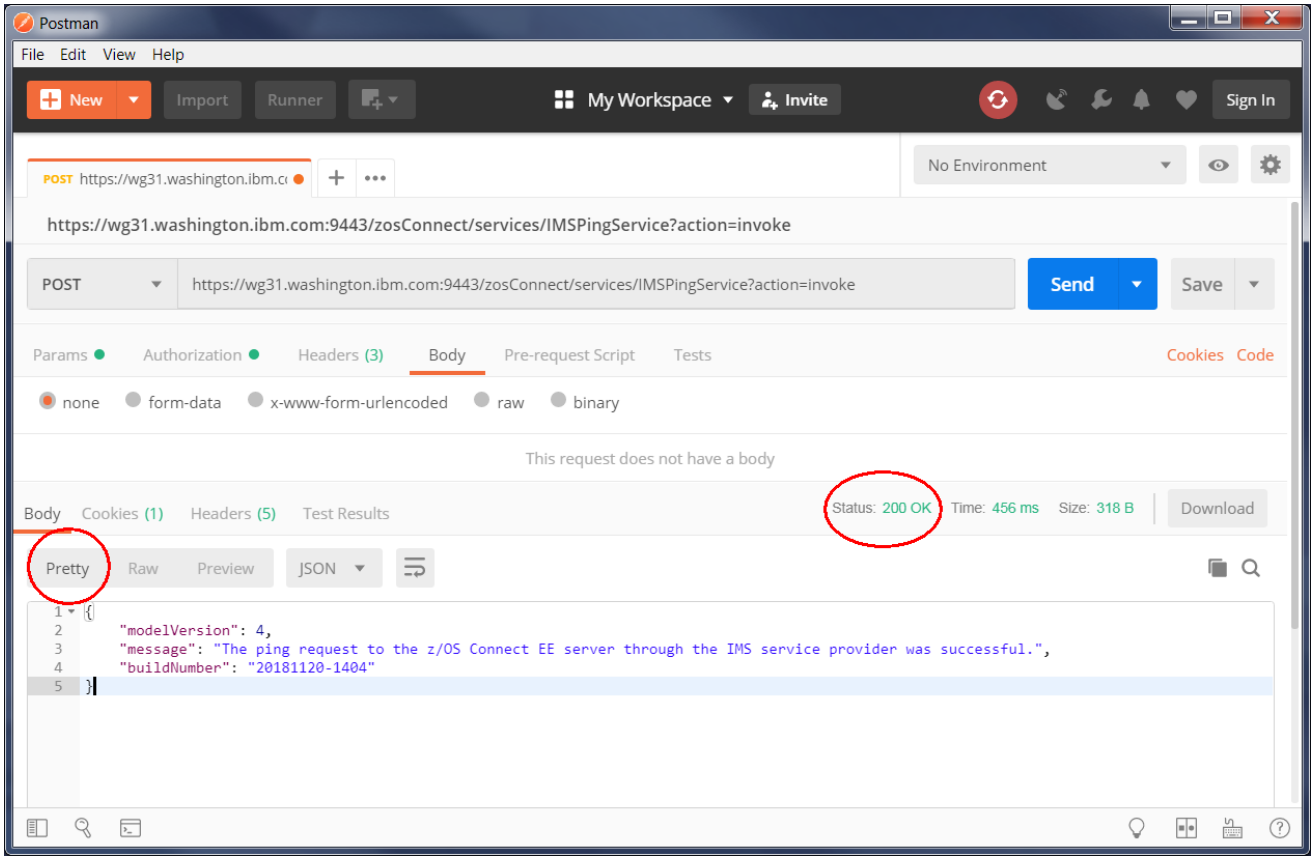


• Next select the *Headers* tab and under *KEY* use the code assist feature to enter ***Content-Type*** and under *VALUE* use the code assist feature to enter ***application/json***.



Tech-Tip: Code assist simply means that when text is entered in field, all the valid values for that field that match the typed text will be displayed. You can select the desired value for the field from the list displayed and that value will populate that field.

• Next select the *Body* tab and press the S**end** button. Pressing the **Send** button invokes the services. The Status of request should be *200 OK* and pressing the *Pretty* tab will display the response message is an easy to read format, see below.



Using cURL

curl -X POST --user Fred:fredpwd --insecure
https://wg31.washington.ibm.com:9443/zosConnect/services/IMSPingService?action=invoke

```
curl -X POST --user Fred:fredpwd --insecure
https://wg31.washington.ibm.com:9453/zosConnect/services/IMSPingService?action=invoke
{"modelVersion":4,"message":"The ping request to the z\/OS Connect EE server through
the IMS service provider was successful.","buildNumber":"20181120-1404"}
```

Tech-Tip: In the above example:

--user Fred:fredpwd could have been specified as ***--header "Authorization: Basic RnJlZDpmcmVkcHdk"***

--insecure is a *cURL* directive that tells *cURL* to ignore the self-signed certificate sent by the z/OS Connect EE server

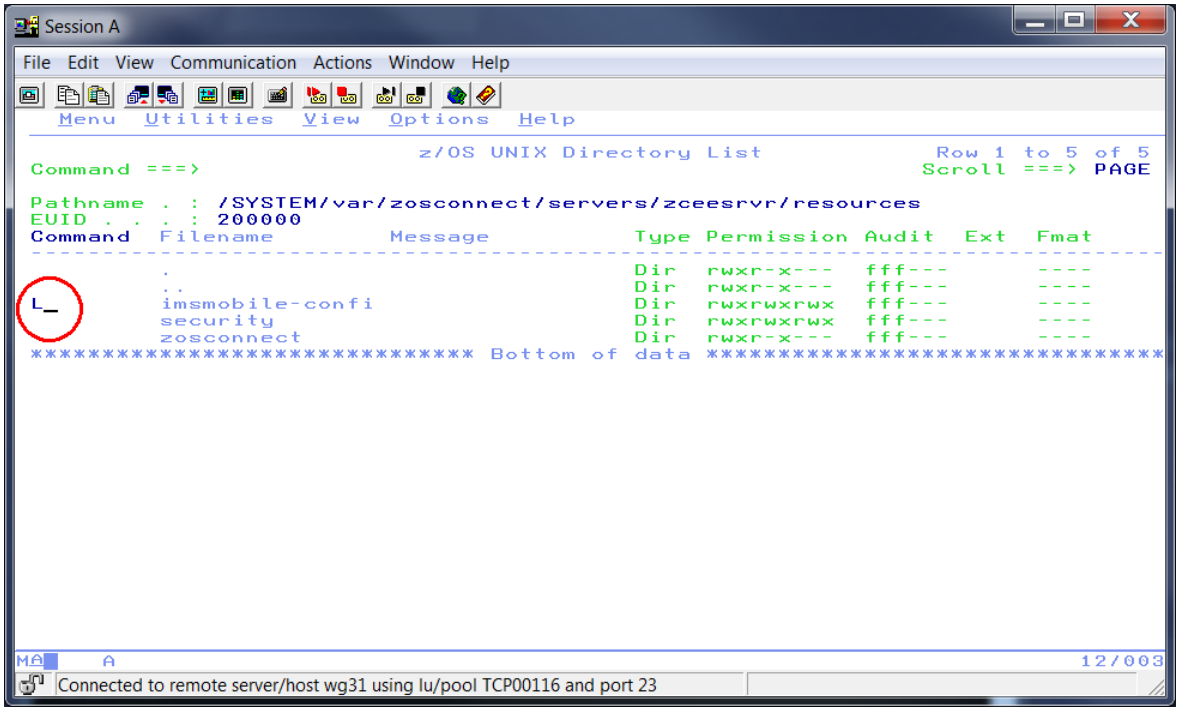
The text in green is the JSON response message.

IMS definitions (connections and interactions)

In this section you will update the IMS Connection information in your z/OS Connect EE server by adding information required to access IMS Connect and the IMS region..

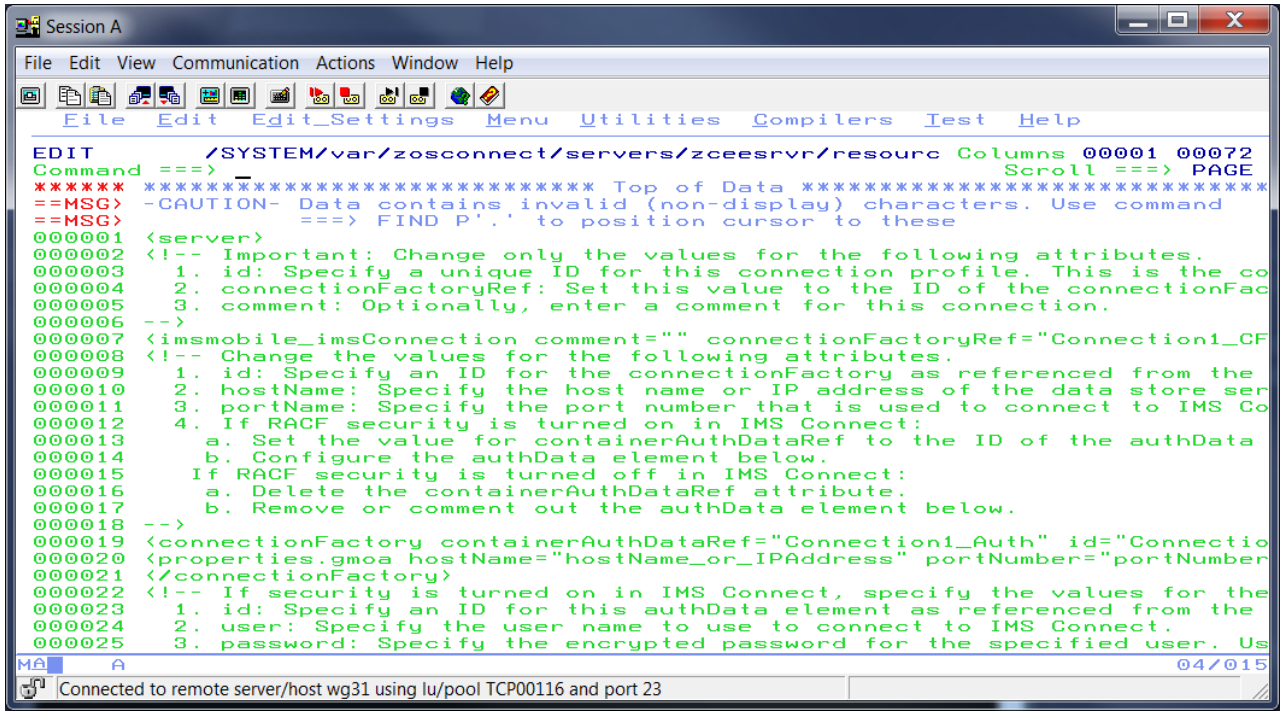
 In an ISPF session go to ISPF option 3.4 (*Data Set List Utility*) and enter ***/var/zosconnect/servers/zceesrvr/resources*** in the area beside *Dsname Level* and press **Enter**.

 On the *z/OS UNIX Directory List* panel enter an ***L*** beside the *imsmobile-config* directory and press **Enter**.



 This will display a list of 4 subdirectories. The contents of subdirectories *connections* and *interactions* need to be updated.

 Enter ***L*** beside *connections* and press **Enter**. Enter ***EA*** beside file *ims-connections.xml* to open this file using the Ascii editor.



 Make the following changes

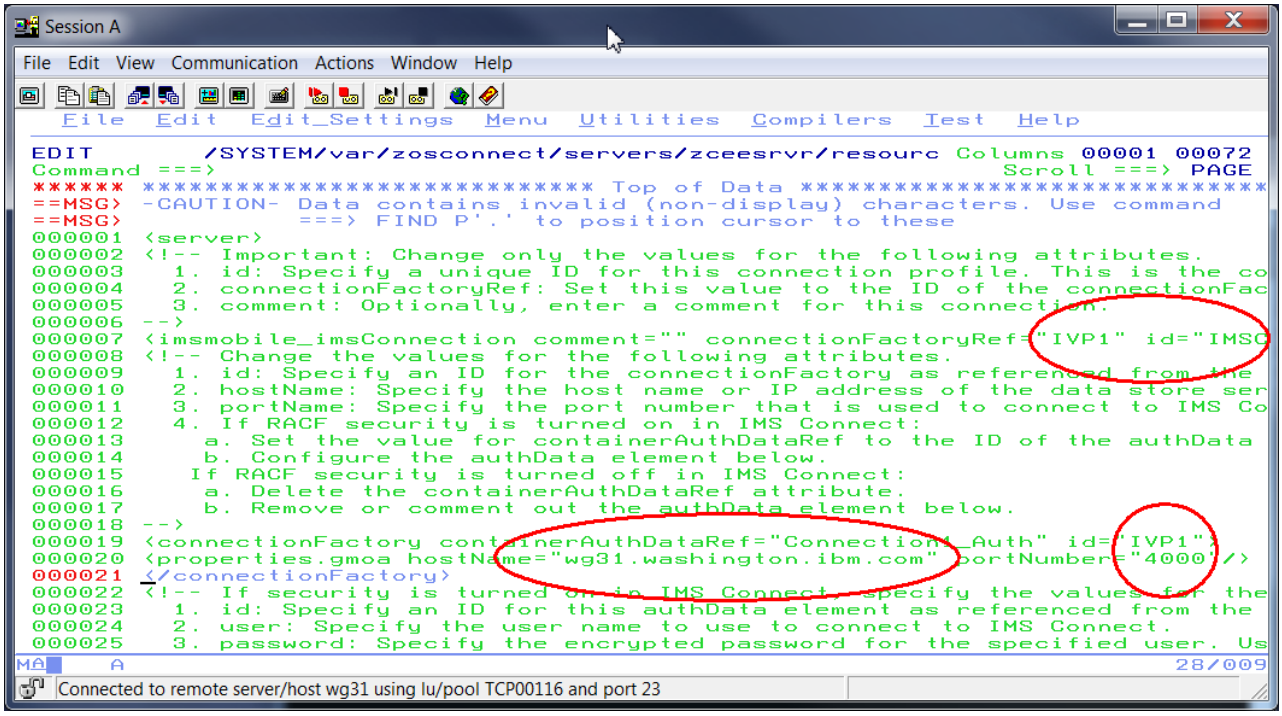
- For *imsmobile_imsConnection* change the value of the *connectionFactoryRef* attribute from *Connection1_CF* to ***IVP1*** and value of the *id* attribute from *Connection1* to ***IMSCONN*** (you may have to scroll to the right to enter IMSCONN.
- For *connectionFactory* change the value of the *id* attribute from *Connection1_CF* to ***IVP1***.
- For *properties.gmoa* change value of *hostname* attribute from *hostName_or_IPAddress* to ***wg31.washington.ibm.com*** and the value of *portNumber* attribute from *portNumber* to ***4000*** as shown below.

Note, the IMS Connect is configured to not use RACF so no changes are required for the *authData* element. Also, password can be stored encrypted as per the comment about the *secureUtility* command.

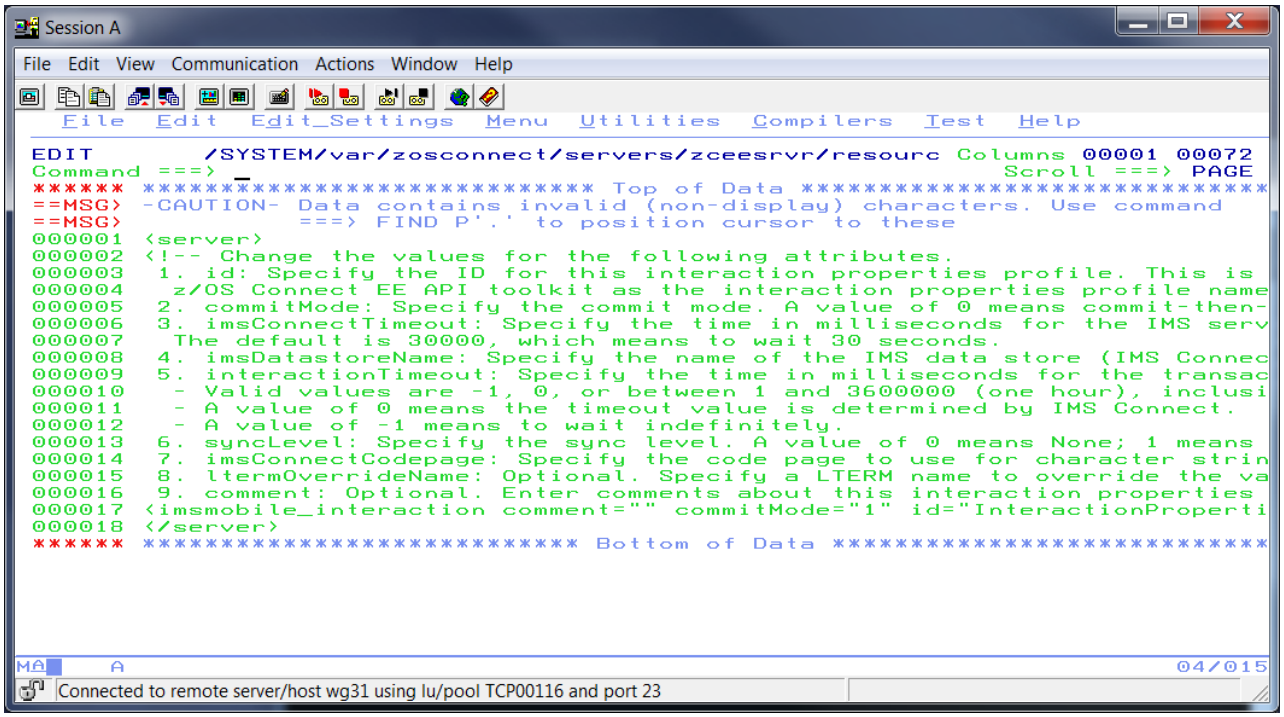
Tech Tip: The port number is obtained from the PORTID parameter configured for the IMS Comment task.

```
HWS=(ID=IMS14HWS,XIBAREA=100,RACF=N,RRS=N)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=SYSSTC,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,TMEMBER=OTMAMEM)
```

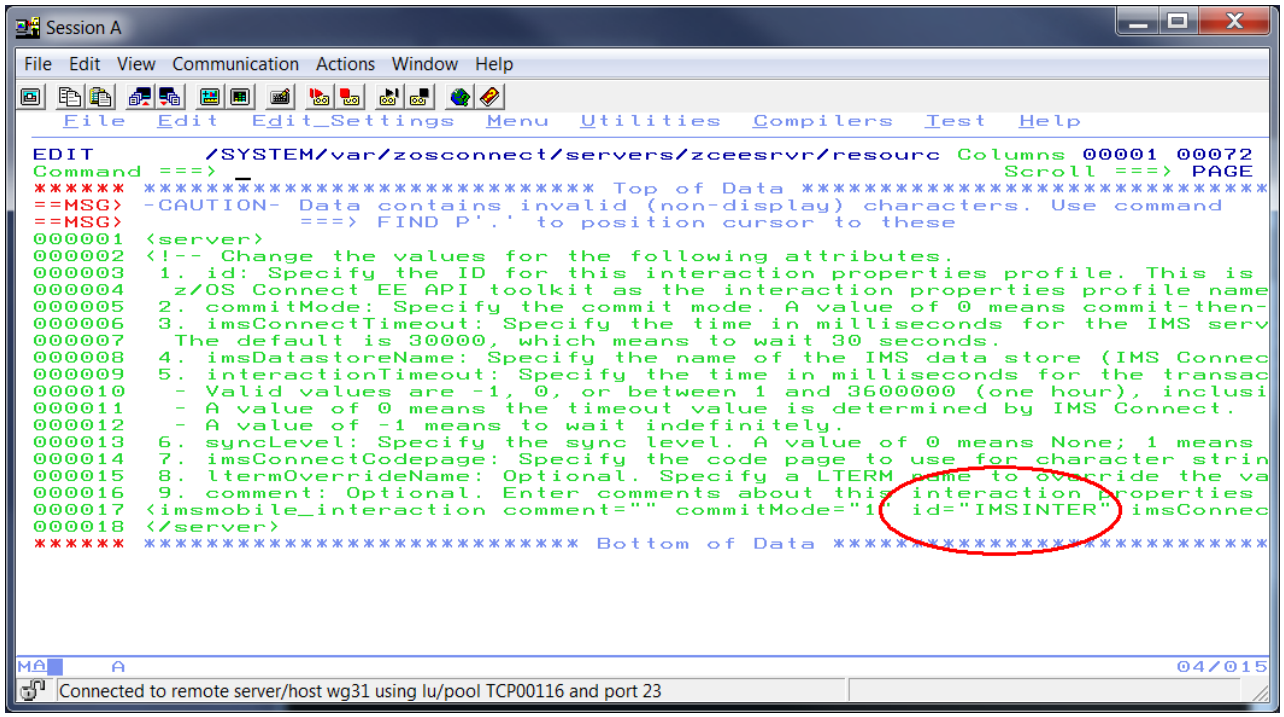
□ Exit the editor and save the changes.



□ Exit back to the list of subdirectories and place an ***L*** beside *interactions.xml* and press **Enter** to open this file using the Ascii editor.



 In the *ims-interactions.xml* scroll to the right and change the value of *imsDatastoreName* attribute from *IMS1* to ***IVP1*** (to match the *DATASTORE* **ID** configured for IMS Connect, e.g. IVP1) and change the value of the *id* attribute from *InteractionProperties1* to ***IMSINTER***.



Developing RESTful Services for IMS

Once the IMS OTMA configuration is completed follow the instructions for the development and deployment of services in the ***Developing RESTful APIs for IMS Transactions*** document at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop. This document shows how to develop and deploy IMS services as well as showing how to develop and deploy APIs that consume these services. For the purposes of this document we are only interested in deploying and testing services, but feel free to develop and test APIs also.

Test the Services

If you have followed the instructions in ***Developing RESTful APIs for IMS Transactions***, you should have a service named *ivtnoService* deployed to the server. This service can be used to test connectivity to IMS from the z/OS Connect server. The service and infrastructure should be tested before developing an API to ensure the infrastructure and the request and response messages are as expected.

- Follow the instructions for testing services in either section *Testing z/OS Connect Services Using Postman* on page 68 or section *Testing z/OS Connect Services Using cURL* on page 74 to test the *ivtnoService* service.

- For the *ivtnoService* service use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/ivtnoService?action=invoke*

  - To display a phone book contact use JSON request message:

```
{
  "INPUT_MSG": {
      "IN_COMMAND": "DISPLAY",
      "IN_LAST_NAME": "LAST1"
   }
}
```

With expected JSON response message:

```
{"OUTPUT_AREA":{"OUT_ZIP_CODE":"D01\/R01","OUT_FIRST_NAME":"FIRST1","OU
T_EXTENSION":"8-111-1111","OUT_MESSAGE":"ENTRY WAS
DISPLAYED","OUT_LAST_NAME":"LAST1"}}
```

To delete a phone book contact use JSON request message:

```
{
  "INPUT_MSG": {
      "IN_COMMAND": "DELETE",
      "IN_LAST_NAME": "LAST1"
   }
}
```

With expected JSON response message:

```
{"OUTPUT_AREA":{"OUT_ZIP_CODE":"D01\/R01","OUT_FIRST_NAME":"FIRST1","OUT_EXTENSI
ON":"8-111-1111","OUT_MESSAGE":"ENTRY WAS DELETED","OUT_LAST_NAME":"LAST1"}}
```

To add a phone book contact use JSON request message:

```
{
  "INPUT_MSG": {
      "IN_COMMAND": "ADD",
      "IN_LAST_NAME": "LASTZ",
      "IN_FIRST_NAME": "FIRSTZ",
      "IN_EXTENSION": "0065",
      "IN_ZIP_CODE": "8000000"
   }
}
```

With expected JSON response message:

```
{"OUTPUT_AREA":{"OUT_ZIP_CODE":"8000000","OUT_FIRST_NAME":"FIRSTZ","OUT_EXTENSION":
"0065","OUT_MESSAGE":"ENTRY WAS ADDED","OUT_LAST_NAME":"LASTZ"}}
```

To update a phone book contact use JSON request message:

```
{
  "INPUT_MSG": {
      "IN_COMMAND": "ADD",
      "IN_LAST_NAME": "LASTZ",
      "IN_FIRST_NAME": "FIRSTZ",
      "IN_EXTENSION": "0065",
      "IN_ZIP_CODE": "8111111"
   }
}
```

With expected JSON response message:

```
{"OUTPUT_AREA":{"OUT_ZIP_CODE":"8111111","OUT_FIRST_NAME":"FIRSTZ","OUT_EXTENSION":
"0065","OUT_MESSAGE":"ENTRY WAS UPDATED","OUT_LAST_NAME":"LASTZ"}}
```

If these tests complete as expected, then the server can communicate with IMS and the infrastructure is ready for the deployment of APIs. The development, deployment and testing of APIs can proceed.

Db2 RESTful APIs

Accessing Db2 from z/OS Connect EE differs from the way z/OS Connect EE accesses other z/OS subsystems. The other subsystems are accessed by using standard subsystem interfaces (e.g., OTMA, IPIC, JMS, etc.). A z/OS Connect EE server accesses Db2 not as a Db2 client using JDBC but rather as a RESTful client accessing a Db2 native REST service.

This may raise the question as to what value-add does z/OS Connect EE provide if a Db2 native REST services are still required for z/OS Connect EE. The answer is that (1) the Rest services support provided by Db2 only supports the POST method with only a few administrative services that support the GET method. There is no support for PUT or DELETE methods normally expected for a robust RESTful service. Another reason (2) is that the API function of transforming JSON request or response messages, e.g. assigning values or removing fields from the interface is not available when using the Db2 native REST services directly. A Swagger document (3) used for integration into API management products or development tools is available from z/OS Connect EE whereas Db2 only provides a JSON document describing its service. If a full function RESTful API with support for the major HTTP methods (POST, PUT, GET and DELETE) and transforming JSON payloads and generating a Swagger document is required then z/OS Connect EE is the solution. Finally (4), Db2 native REST services seems to only support basic authentication. Adding a z/OS Connect EE server in front of Db2 provides support for third party authentication tokens and asserting identities using client certificates.

User RESTful services for Db2 are defined either using a Db2 provided RESTful administrative service(*Db2ServiceManager)* or by using the Db2 BIND command using an update provided in Db2 PTF UI51748 for V12 and UI51795 for V11.

Creating Db2 REST Services

Review the job below. Submitting this job for execution will define a Db2 native REST service that selects a single row from table USER1.EMPLOYEE (see below) based on the employee number (column EMPNO).

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.Db2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.Db2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT  DD *
SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName",
  MIDINIT AS "middleInitial", LASTNAME as "lastName",
   WORKDEPT AS "department", PHONENO AS "phoneNumber",
   JOB AS "job"
 FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN  DD *
 DSN SYSTEM(DSN2)

 BIND SERVICE(SYSIBMSERVICE) -
  NAME("selectEmployee") -
  SQLENCODING(1047) -
  DESCRIPTION('Select an employee from table USER1.EMPLOYEE')
/*
```

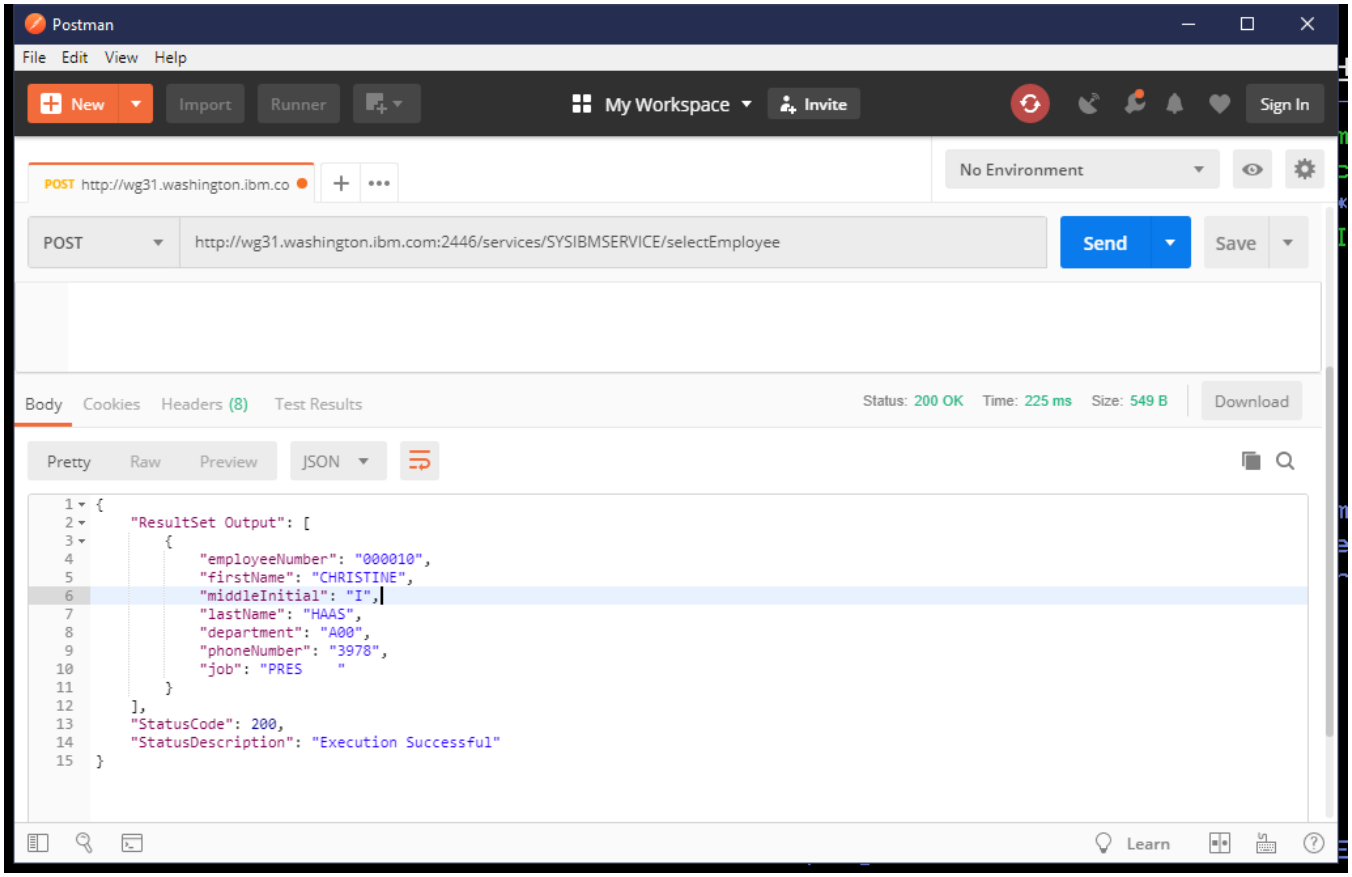
```
CREATE TABLE USER1.EMPLOYEE
              (EMPNO     CHAR(6)        NOT NULL,
               FIRSTNME  VARCHAR(12)    NOT NULL,
               MIDINIT   CHAR(1)        NOT NULL,
               LASTNAME  VARCHAR(15)    NOT NULL,
               WORKDEPT  CHAR(3)                ,
               PHONENO   CHAR(4)                ,
               HIREDATE  DATE                   ,
               JOB       CHAR(8)                ,
               EDLEVEL   SMALLINT               ,
               SEX       CHAR(1)                ,
               BIRTHDATE DATE                   ,
               SALARY    DECIMAL(9, 2)          ,
               BONUS     DECIMAL(9, 2)          ,
               COMM      DECIMAL(9, 2)          ,
               PRIMARY KEY(EMPNO));
```

Tech Tip: To delete a service created by using the Db2 BIND command use the Db2 FREE command, e.g. FREE SERVICE(SYSIBMSERVICE."selectEmployee")

The *selectEmployee* Db2 native REST service can be tested with *Postman* or *cURL* with URL ***http://wg31.washington.ibm.com:2446/services/selectEmployee***
and JSON request message.

```
{
      "employeeNumber": "000010"
}
```

- Using Postman



- Using cURL:

curl -X POST --user USER1:USER1 --header "Content-Type: application/json" -d @selectEmployee.json ***http://wg31.washington.ibm.com:2446/services/selectEmployee***

{"ResultSet Output":[{"employeeNumber":"000010","firstName":"CHRISTINE","middleInitial":"I","lastName": "HAAS","department":"A00","phoneNumber":"3978","job":"PRES "}],"StatusCode": 200,"StatusDescription": "Execution Successful"}

Other Db2 native REST services can be created using the same JCL but with different input for the DSNSTMT DD statement. A service that deletes a row from a table , a service that selects a row based on columns *department* and *job*, a service that adds a row, a service that updates an existing and finally a service that can display all the columns of a row can be created using the SQL statements below:

- DELETE FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
- SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName", MIDINIT AS "middleInitial", LASTNAME as "lastName", WORKDEPT AS "department", PHONENO AS "phoneNumber", JOB AS "job"
  FROM USER1.EMPLOYEE WHERE JOB = :job AND WORKDEPT = :department
- INSERT INTO USER1.EMPLOYEE
  (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,PHONENO,
  HIREDATE,JOB,EDLEVEL,SEX,BIRTHDATE,SALARY,BONUS,COMM)
  VALUES (:employeeNumber, :firstName, :middleInitial, :lastName,:department,
  :phoneNumber, :hireDate, :job,:educationLevel, :sex, :birthDate,:salary, :bonus, :commission)
- UPDATE USER1.EMPLOYEE SET SALARY = :salary, BONUS = :bonus, COMM = :commisson
  WHERE EMPNO = :employeeNumber
- SELECT * FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber

Adding Db2 REST support to a z/OS Connect server

Connectivity between the z/OS Connect EE (zCEE) server and a Db2 subsystem is provided by a REST client connection element.

In the sample that will be shown, the Db2 subsystem is running on TCP/IP host *wg31.washington.ibm.com* and its distributed data facility task is listening on port *2446.* The z/OS Connect EE server is running on the same TCP/IP host and is listening on port *9443* for HTTPS requests.

Do the following:

- Go to the server.xml directory, e.g. */var/zosconnect/servers/serverName*
- Edit *server.xml* and add the lines highlighted here in **bold** as shown, see the notes below:

```
<zosconnect_zosConnectServiceRestClientConnection id="db2conn" 1
       host="wg31.washington.ibm.com"    2
       port="2446"  3
       basicAuthRef="dsn2Auth" />  4

 <zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
       userName="USER1"
       password="USER1"/>
```

Notes:

1. This value must match the value that is specified for the *connectionRef* property when a *service* is developed using the z/OS Connect build tool kit.
2. The TCP/IP host name or IP address of the host on which the Db2 subsystem is running.
3. The port assigned to the Db2 DDF task.
4. A reference to an authorization element. Note that the password can be encrypted.

> **Tech Tip:** RACF Passtickets can be used in lieu of basic authentication.

- Save the file.

Developing RESTful Services for Db2 Native REST Services

Once the Db2 configuration is completed follow the instructions for the development and deployment of services in the ***Developing RESTful APIs for Db2 Native Services*** document at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop . The Db2 exercise at this site assumes Db2 APAR PI98649 has been installed on the V11 or V12 Db2 subsystem. It shows how to develop and deploy Db2 services as well as showing how to develop and deploy APIs that consume these services. For the purposes of this document we are only interested in deploying and testing services, but feel free to develop and test APIs also

Test the Services

These services deployed from the above exercise can be used to test connectivity to Db2 from the z/OS Connect server. The service and infrastructure should be tested before developing an API to ensure the infrastructure and the request and response messages are as expected.

Follow the instructions for testing services in either section *Testing z/OS Connect Services Using Postman* on page 68 or section *Testing z/OS Connect Services Using cURL* on page 74 to test the Db2 servicess.

- For service *selectEmployee* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/selectEmployee?action=invoke*
  - and use JSON request message:

```
{
      "employeeNumber": "000010"
}
```

With expected JSON response message:

```
{
    "StatusDescription": "Execution Successful",
    "ResultSet Output": [
        {
            "firstName": "CHRISTINE",
            "lastName": "HAAS",
            "middleInitial": "I",
            "phoneNumber": "3978",
            "department": "A00",
            "job": "PRES    ",
            "employeeNumber": "000010"
        }
    ],
    "StatusCode": 200
}
```

- For service *deleteEmployee* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/selectEmployee?action=invoke* and use JSON request message:

```
{
      "employeeNumber": "000020"
}
```

With expected JSON response message:

```
{"StatusDescription":"Execution Successful",
"Update Count":1,
"StatusCode":200}
```

- For service *selectByRole* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/selectByRole?action=invoke* and use JSON request message:

```
{
      "job": "PRES",
      "department": "A00"
}
```

With expected JSON response message:

```
{
    "StatusDescription": "Execution Successful",
    "ResultSet Output": [
        {
            "firstName": "CHRISTINE",
            "lastName": "HAAS",
            "middleInitial": "I",
            "phoneNumber": "3978",
            "department": "A00",
            "job": "PRES    ",
            "employeeNumber": "000010"
        },
        {
            "firstName": "CHRISTINE",
            "lastName": "HAAS",
            "middleInitial": "I",
            "phoneNumber": "A1A1",
            "department": "A00",
            "job": "PRES    ",
            "employeeNumber": "000011"
        }
    ],
    "StatusCode": 200
}
```

- For service *insertEmployee* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/insertEmployee?action=invoke* and use JSON request message:

```
{
      "employeeNumber": "948478",
      "firstName": "Matt",
      "middleInitial": "T",
      "lastName": "Johnson",
      "department": "A00",
      "phoneNumber": "0065",
      "hireDate": "2013-10-15",
      "job": "staff",
      "educationLevel": "22",
      "sex": "M",
      "birthDate": "1985-06-18",
      "salary": 2000,
      "bonus": 1000,
      "commission": 500
}
```

With expected JSON response message:

```
{
    "StatusDescription": "Execution Successful",
    "Update Count": 1,
    "StatusCode": 200
}
```

- For service *updateEmployee* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/updateEmployee?action=invoke* and use JSON request message:

```
{
      "employeeNumber": "948478",
      "salary": "110000",
      "bonus": "20000",
      "commission": "10000"
}
```

With expected JSON response message:

```
{
    "StatusDescription": "Execution Successful",
    "Update Count": 1,
    "StatusCode": 200
}
```

- For service *displayEmployee* use URL *https://wg31.washington.ibm.com:9443/zosConnect/services/displayEmployee?action=invoke* and use JSON request message:

```
{
      "employeeNumber": “948478”
}
```

With expected JSON response message:

```
{
    "StatusDescription": "Execution Successful",
    "ResultSet Output": [
        {
            "PHONENO": "0065",
            "EDLEVEL": 27,
            "SEX": "M",
            "FIRSTNME": "Matt",
            "MIDINIT": "T",
            "BIRTHDATE": "1985-06-10",
            "SALARY": 110000,
            "COMM": 10000,
            "LASTNAME": "Johnson",
            "WORKDEPT": "A00",
            "HIREDATE": "2003-10-15",
            "BONUS": 20000,
            "EMPNO": "948478",
            "JOB": "staff    "
        }
    ],
    "StatusCode": 200
}
```

IBM MQ RESTful APIs

A new MQ Service Provider was shipped with z/OS Connect EE V3.0.21. The MQ Service Provider shipped with MQ is still supported but users should plan to migrate to the new provider. In the meantime, configuring of the service provider will be covered in this section. Also included in this section is an example of developing and testing a service interface for the MQ one-way service defined in the zCEE server.

Adding MQ Service provider support to a z/OS Connect server

Implementing the MQ Service Provider shipped with z/OS Connect EE requires the addition of a Liberty feature in the *featureManager* element of the *server.xml* file (e.g. feature *zosconnect:mqService-1.0*).

Also require is the location of the JMS provider's (IBM MQ) resource adapter file by using variable *wmqJMSClient.rar.location* and the location of any JMS Provider's executable binaries using variable *nativeLibraryPath* (see below). This resource adapter must be at the V9.0.1 level or later.

```
 <featureManager>
        ….
       <feature>zosconnect:mqService-1.0</feature>
</featureManager>

 <variable name="wmqJmsClient.rar.location"
          value="/usr/lpp/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar"/>
 <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R0M1/java/lib"/>
```

Adding JMS resources to the z/OS Connect EE configuration

The MQ Service Provider is a JMS application and requires the normal Liberty JMS configuration elements.

JMS applications running in Java container requires a *name space* which provides queue manager connection information (*jmsConnectionFactory*) and queue information (*jmsQueue*). This *name space* is accessed when the JMS application does a *Java Naming and Directory Interface* (JNDI) lookup during execution. This *name space* lookup also applies for the MQ Service Provider running in z/OS Connect EE server. For a JMS application running in Liberty the elements required for the *name space* also reside in the server's configuration file. The JMS elements below show the *jmsConnectionFactory* element with the attributes required to connect to the target queue manager and three *jmsQueue* elements with the attributes required to access 3 queues defined in that queue manager.

```
<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
         connectionManagerRef="ConMgr1">
         <properties.wmqJMS transportType="BINDINGS"
                  queueManager="MQS1" />
</jmsConnectionFactory>

<jmsQueue id="q1" jndiName="jms/default">
       <properties.wmqJms
           baseQueueName="ZCONN2.DEFAULT.MQZCEE.QUEUE"
           targetClient="MQ"
           CCSID="37"/>
</jmsQueue>
```

 The jmsConnectionFactory element associates the JMS connection factory (jndiName) with the target queue manager and details on how to connect to this queue manager.

 The jmsQueue elements provide details that associate the JMS destination (jndiName) with the target queue (baseQueueName) and its MQ JMS properties. In particular the MQ JMS property of CCSID=37 was added to ensure the message would be converted to EBCDIC and the targetClient property was added to indicate that no MQRFH2 header was to be included (the target application is an MQI application which does not expect an MQRFH2 header).

Developing RESTful Services for MQ

Once the MQ service provider configuration is completed follow the instructions for the development and deployment of services in the ***Developing RESTful APIs for MQ (zCEE MQ provider)*** document at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop. This document shows how to develop and deploy MQ services as well as showing how to develop and deploy APIs that consume these services. For the purposes of this document we are only interested in deploying and testing services, but feel free to develop and test APIs also.

Test the Services

If you have followed the instructions in ***Developing RESTful APIs for MQ Services*** you should have a service named *mqPut* deployed to the server. This service can be used to test connectivity to an MQ queue manager from the z/OS Connect server. The service and infrastructure should be tested before developing an API to ensure the infrastructure and the request and response messages are as expected.

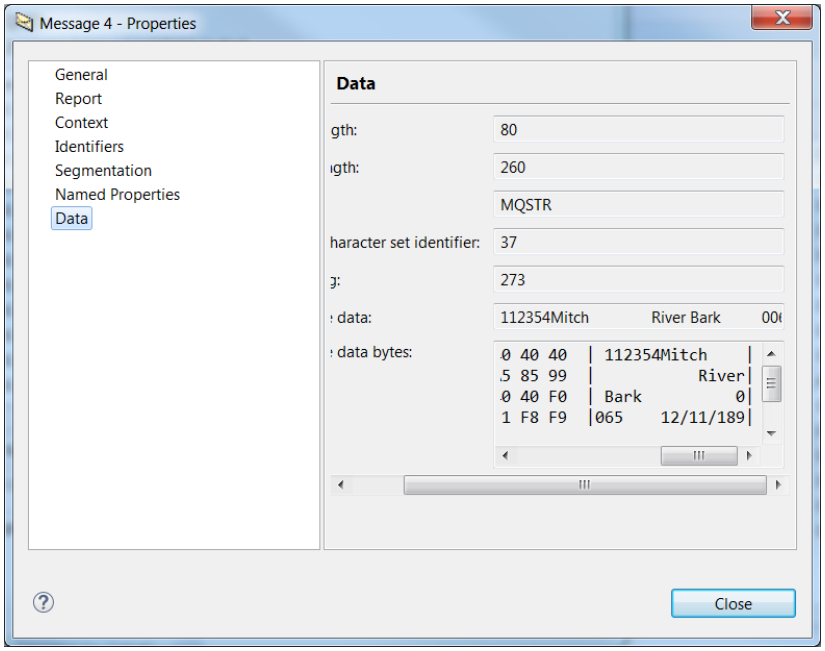
 Follow the instructions for testing services in either section *Testing z/OS Connect Services Using Postman* on page 68 or section *Testing z/OS Connect Services Using cURL* on page 74 to test the *FileaQueue* service.

- For the *mqPut* service use URL
https://wg31.washington.ibm.com:9443/zosConnect/services/mqPut?action=invoke

  - To put a message on a queue use JSON request message:

```
{
  "MQPUTOperation": {
    "mqmessage": {
      "stat": " ",
      "numb": "112354",
      "name": "Mitch",
      "addrx": "River Bark",
      "phone": "0065",
      "datex": "12/11/18 ",
      "amount": "948478",
      "comment": ""
    }
  }
}
```

The request should succeed with a *204 No Content* response. No JSON response message is expected but the messages should show up on the queue.



If this test complete as expected, then the server can communicate with the queue manager and the infrastructure is ready for the deployment of APIs. The development, deployment and testing of APIs can proceed.

Advanced Topics

Testing z/OS Connect Services Using Postman

Two products which seem to be most popular tools for testing RESTful APIs can be used to test the services gererated by z/OS Connect tooling. The two products are *Postman* which is available for downloading from https://www.getpostman.com/apps and *cURL (client URL)* which is available for downloading from https://curl.haxx.se/download.html. The use of Postman will be shown in this section.

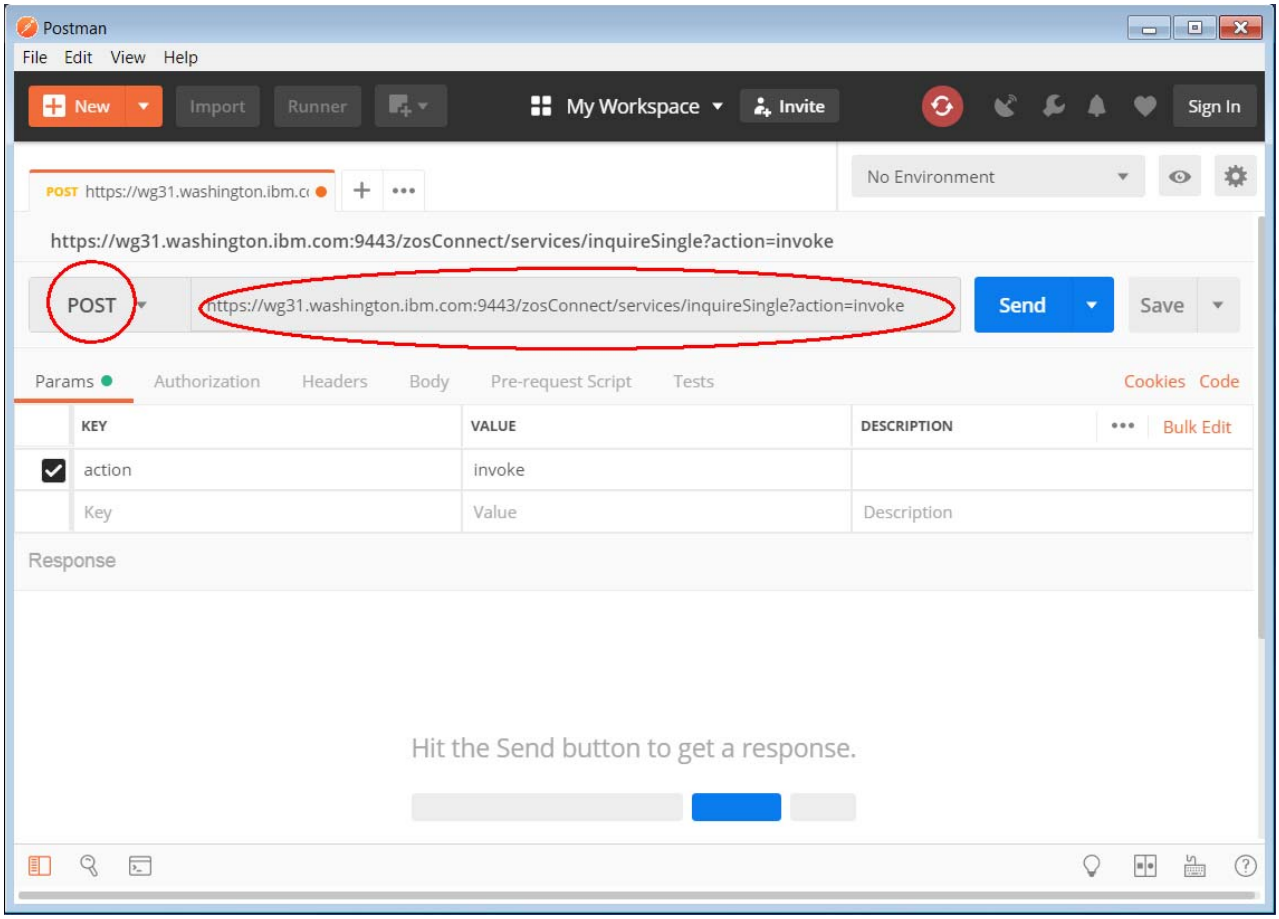
The basic steps shown here apply for any z/OS Connect services, not just for CICS service shown here.

- Every REST request will be a *POST* method
- Every service will include *?action=invoke* attribute as part of the service name
- Every request will require a basic authorization token
- Every request will specify *Content-Type* of *application/json*
- The only items that vary are the service name and the request and response JSON messages

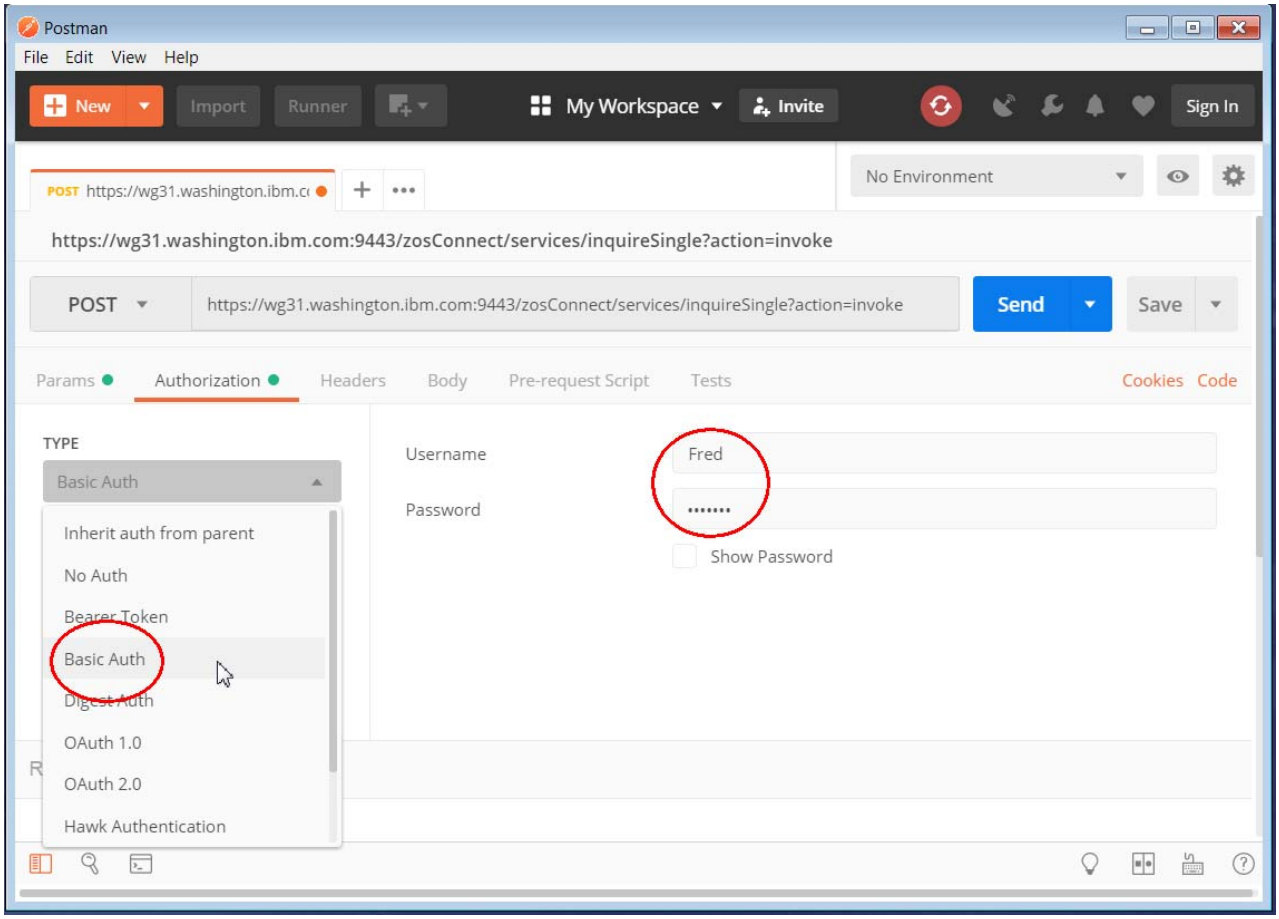
Using Postman

- To test the *inqurieSingle* service open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages, use the down arrow to select **POST** and enter in the URL area containing an invoke request the service name (see below).

https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?action=invoke

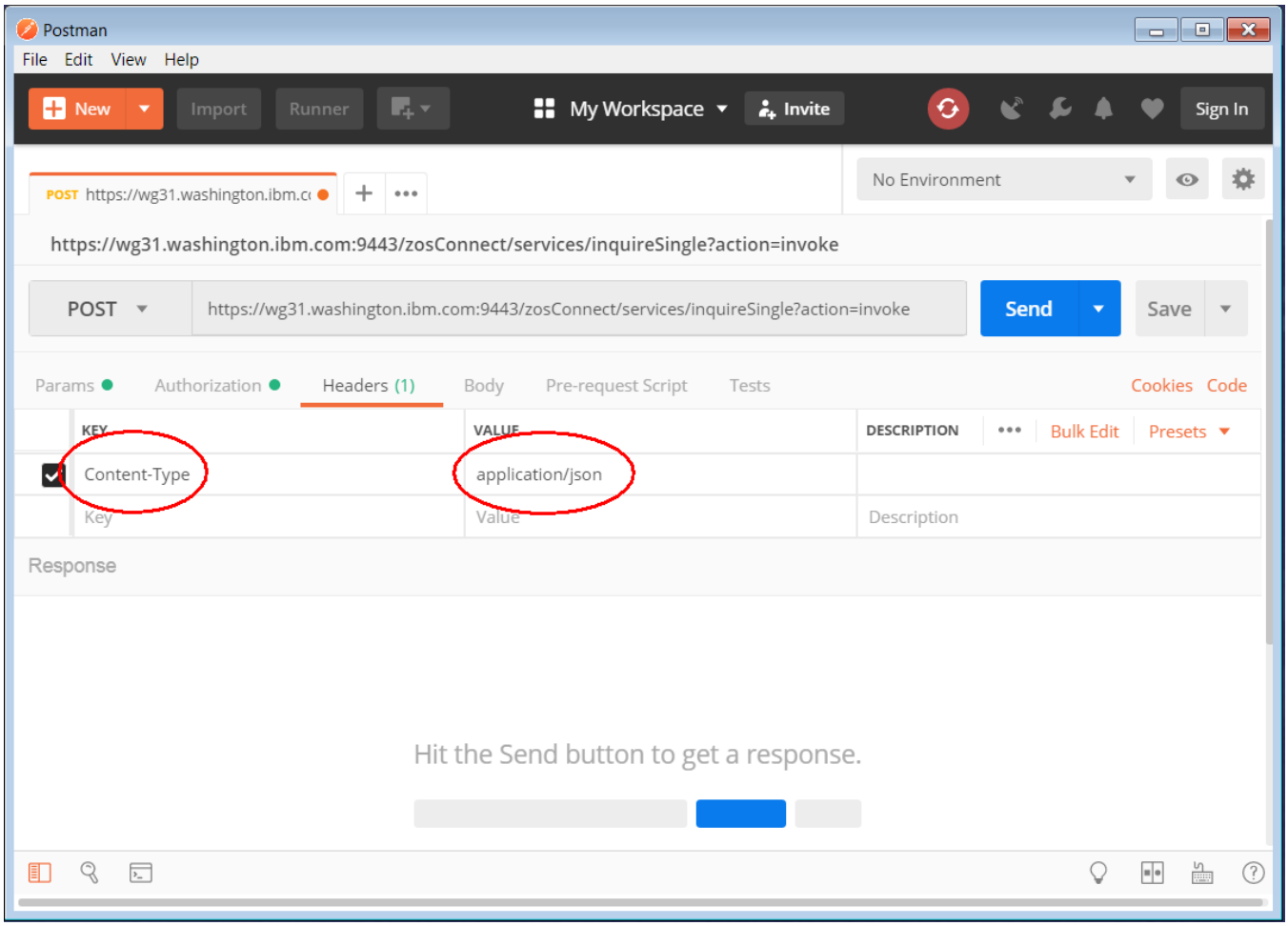


• No *query* or *path* parameters are required so next select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter ***Fred*** as the username and ***fredpwd*** as the Password (these are the identity and password defined in the server.xml).



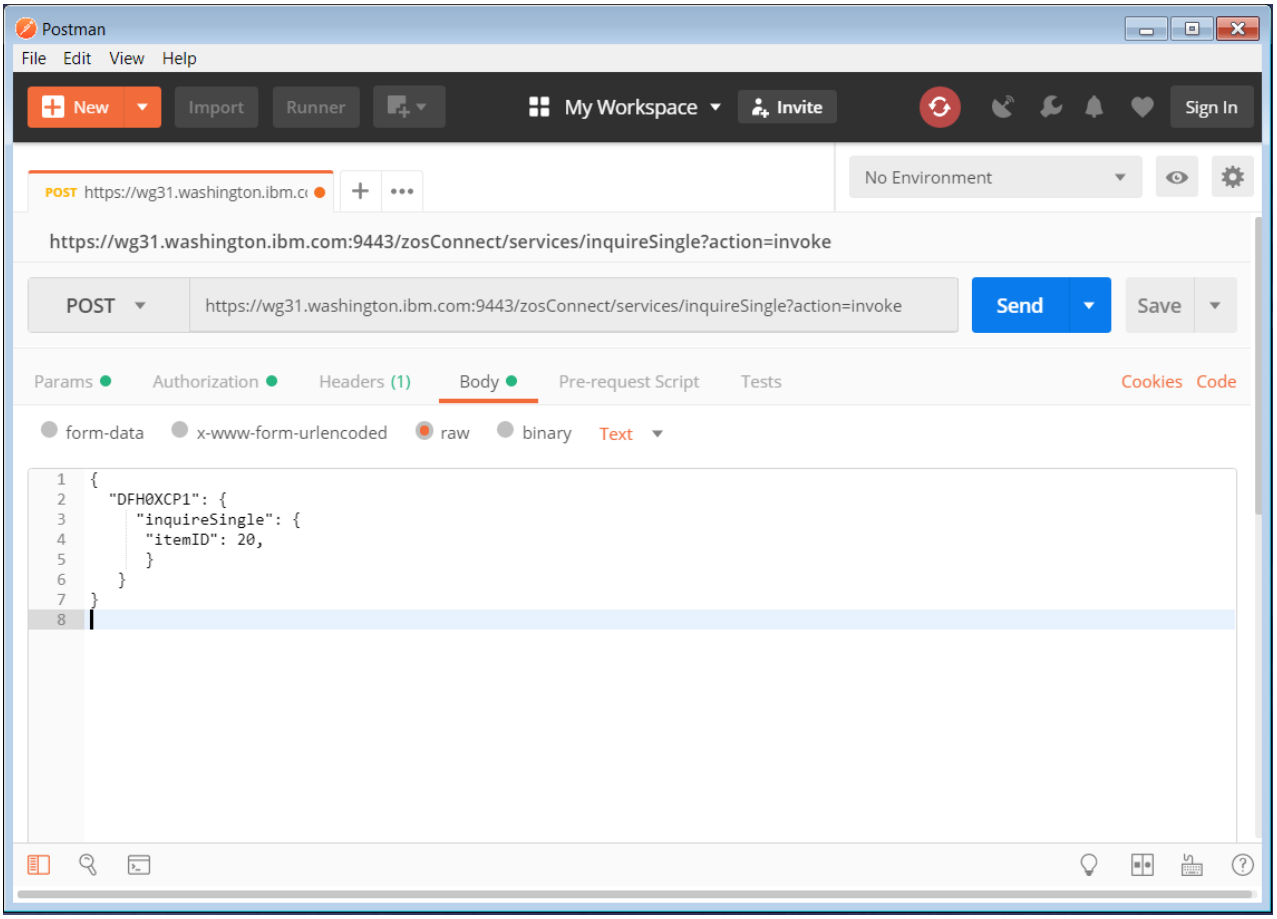
• Next select the *Headers* tab and under *KEY* use the code assist feature to enter ***Content-Type*** and under *VALUE* use the code assist feature to enter ***application/json***.

Tech-Tip: Code assist simply means that when text is entered in field, all the valid values for that field that match the typed text will be displayed. You can select the desired value for the field from the list displayed and that value will populate that field.

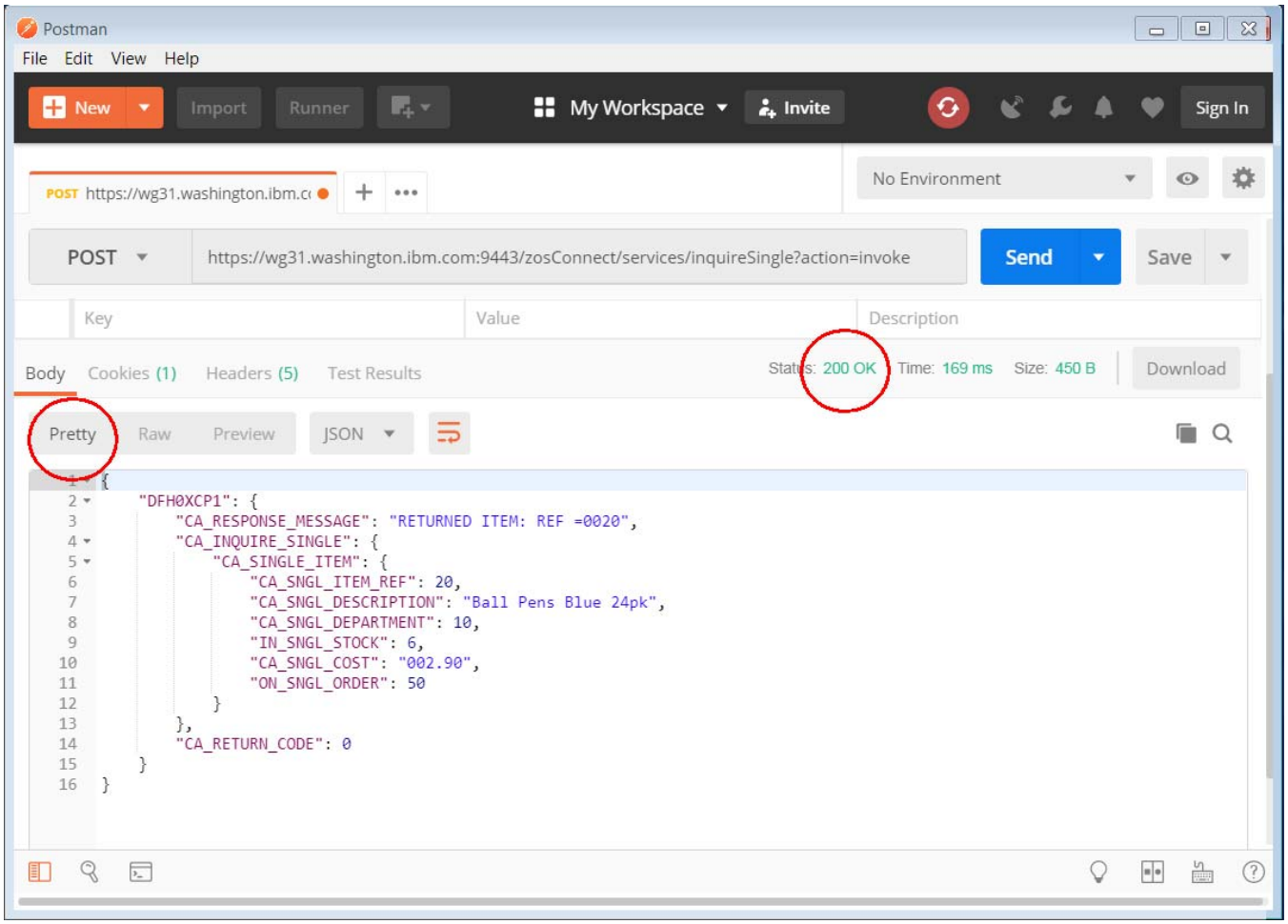


• Next select the *Body* tab and select the *raw* radio button and enter the JSON message below in the *Body* area and press the S**end** button.

```
{
  "DFH0XCP1": {
     "inquireSingle": {
      "itemID": 20,
      }
   }
}
```



• Pressing the **Send** button invokes the API. The Status of request should be *200 OK* and pressing the *Pretty* tab will display the response message is an easy to read format, see below.



Testing z/OS Connect Services Using cURL

Two products which seem to be most popular tools for testing RESTful APIs can be used to test the services gererated by z/OS Connect tooling. The two products are *Postman* which is available for downloading from https://www.getpostman.com/apps and *cURL (client URL)* which is available for downloading from https://curl.haxx.se/download.html. The use of cURL will be shown in this section.

The basic steps shown here apply for any z/OS Connect services, not just for CICS service shown here.

- Every REST request will be a *POST* method
- Every service will include *?action=invoke* attribute as part of the service name
- Every request will require a basic authorization token
- Every request will specify *Content-Type* of *application/json*
- Every request will contain an -d attribute which specifies a file contain the JSON request message
- The only items that vary are the service name and the request and response JSON messages

Using cURL

The *cURL* tool provides a command line interface to REST APIs. The same service just tested with *Postman* can be tested with *cURL* as shown here.

- Enter the command below at the command prompt

```
curl -X POST --user Fred:fredpwd --header "Content-Type: application/json"
-d @inquireSingle.json --insecure
https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?action=invoke
{"DFH0XCP1":{"CA_RESPONSE_MESSAGE":"RETURNED ITEM: REF
=0020","CA_INQUIRE_SINGLE”:{"CA_SINGLE_ITEM":{"CA_SNGL_ITEM_REF":20,"CA_SNGL_DESCRIPT
ION":"Ball Pens Blue 24pk", "CA_SNGL_DEPARTMENT":10, "IN_SNGL_STOCK":6,
“CA SNGL COST":"002.90", "ON SNGL ORDER":50}}, "CA RETURN CODE":0}}
```

Tech-Tip: In the above example:

--user Fred:fredpwd could have been specified as ***--header "Authorization: Basic RnJlZDpmcmVkcHdk"***

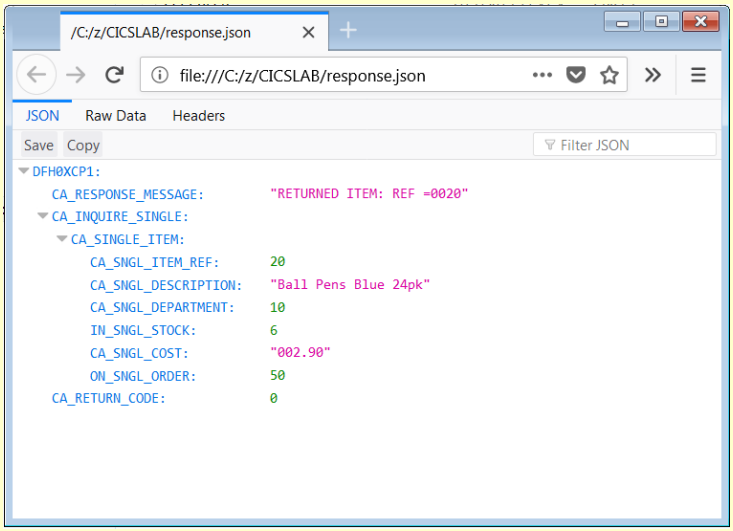
@inquireSingle.json is a file in the same directory that contains the request JSON message

--insecure is a *cURL* directive that tells *cURL* to ignore the self-signed certificate sent by the z/OS Connect EE server

The text in green is the JSON response message.

Tech-Tip: Another useful cURL directive is *-o response.json*

When this directive is used the JSON response message is written to a file named *response.json* which then can be opened with Firefox and viewed in a more readable format, e.g. command *firefox response.json*



Entering Firefox as a command assumes the directory containing the Firefox executable has been added to the PATH environment variable.

Tech-Tip: A recent update of Windows included an update to curl.exe file in the *c:\Windows\System32* directory. This update broke my use of cURL when trying to do SSL handshakes. I was receiving messages

curl: (77) schannel: next InitializeSecurityContext failed: SEC_E_UNTRUSTED_ROOT (0x80090325) - The certificate chain was issued by an authority that is not trusted.

The resolution to this problem was to place the directory which contained the curl.exe I wanted to use earlier in the PATH environment variable than the Window's version of the curl.exe.

WOLA Security

WOLA connections between z/OS Connect EE servers and CICS, MVS batch or other subsystems use CBIND RACF resources to provide security. For example, if the following zosLocalAdapters element was define in the server.xml

```
<zosLocalAdapters
    wolaGroup="MYSERVER"
    wolaName2="MYSERVER"
    wolaName3="MYSERVER" />
```

Then the following RACF would be required

Grants an ID general access to WOLA interface to the RACF identities of a CICS region and MVS batch job or task

```
RDEF CBIND BBG.WOLA.MYSERVER.MYSERVER.MYSERVER UACC(NONE)OWNER(SYS1)
PERMIT BBG.WOLA.MYSERVER.MYSERVER.MYSERVER CLASS(CBIND) ACCESS(READ) ID(cics_id,mvs_id)
```

Beyond the simple server.xml security elements

Turning off SSL and Authentication

By default, z/OS Connect EE will require both transport security (commonly referred to as "SSL," but more precisely called "TLS," or Transport Layer Security) and authentication.

Earlier in this document we saw that requirement surface: the instructions had you accept the security challenge caused by the self-signed server certificate, and then supply the userid and password.

But you may have certain services or APIs on which you do not wish to enforce transport security or authentication. z/OS Connect EE provides a way to turn off either or both.

No Security?Yes, there are use-cases where transport security or authentication is not needed:

When z/OS Connect EE is inside the network secure zone, behind firewalls and authentication devices. In that case you may decide the overhead of transport encryption is not needed. And you may decide that authentication at a mid-tier device is sufficient and z/OS Connect EE can trust the traffic that flows back from there.

The service being exposed is of such low importance that encryption or authentication is not required. An example of this is a service that provides the day's menu in the office cafeteria.

If you deem encryption and/or authentication unnecessary, you can turn it off at the API or service level.

Turning off at the API level

In the CICS section of this document we illustrated the deployment of the catalog API, with the API defined in the server.xml with this:

```
<zosconnect_zosConnectAPIs location="">
    <zosConnectAPI name="catalog" />
</zosconnect_zosConnectAPIs>
```

In that case the API would require, *by default*, both transport security and authentication.

You could turn both off with:

```
<zosconnect_zosConnectAPIs location="">
    <zosConnectAPI name="catalog"
     requireAuth="false"
     requireSecure="false" />
</zosconnect_zosConnectAPIs>
```

Where *requireAuth* controls authentication, and *requireSecure* controls transport layer encryption. Coding ***"false"*** turns off the requirement for the API.

Clients may then access this API without authenticating and without going through the handshake protocol to establish encryption. This is true even if the underlying service definition still requires both authentication and encryption.

Turning off at the service level

Alternatively, you can turn off the authentication and transport security at the service:

```
<zosconnect_zosConnectService id="inquireSingleService"
      requireAuth="false"
      requireSecure="false"
      serviceName="inquireSingle"
      dataXformRef="xformJSON2Byte"
      serviceDescription="Inquire on an item in the catalog"
      serviceRef="catalog" />
```

The following is from the Knowledge Center.

> **Note**
>
> If your service is called as part of an API call, the interceptors and security configuration included with the API will override the configuration included in the service.

Common Problems … Symptoms and Causes

In this section we will provide a catalog of common problems we have seen and provide information to identify the problem and correct.

Note: This list is *not* exhaustive. We will add things to this as we come across them.

"Angel process not compatible with local communication service"

If you see this error:

```
CWWKB0307E: The angel process on this system is not compatible with the local communication service. The current angel version is 2, but the required angel version is 3
```

It is because the Angel process started task in use by your Liberty z/OS server is running at a code level below what's required for z/OS Connect EE.

This issue can arise when you have an existing Angel process – for example, used by z/OSMF with z/OS 2.1 or higher – and you intend to re-use that Angel for your z/OS Connect EE instances. However, if that existing Angel is operating at a lower code level, you will see the message illustrated above.

The corrective action is to stop the Angel, update the Angel's JCL start procedure and point the SET ROOT variable to the installation path for z/OS Connect EE (or any Liberty z/OS installation at 8.5.5.8 or above), and restart the Angel:

```
//BBGZANGL PROC PARMS='',COLD=N,NAME=''
//*-------------------------------------------------------------------
// SET ROOT='<path to Liberty installation>'
//*-------------------------------------------------------------------
//* Start the Liberty angel process
//*-------------------------------------------------------------------
//STEP1   EXEC PGM=BPXBATA2,REGION=0M,
//  PARM='PGM &ROOT./lib/native/zos/s390x/bbgzangl COLD=&COLD NAME=X
//         &NAME &PARMS'
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
```

Note: Stopping the Angel on an LPAR implies all Liberty z/OS instances on the LPAR must come down. Schedule this update during a maintenance window.

Abend S138 - WOLA three-part name not unique on the system

To use WOLA, the server.xml must include the "three-part name" used when external address spaces WOLA-register into the Liberty z/OS server.

When the Liberty z/OS server starts, that three-part name is checked against a list of other three-part names in use on the system. (The list is maintained by the Angel process.)

The three-part name must be unique on the LPAR. If it is not, the server will not start and you will experience an S138 abend:

```
CEE3250C The system or user abend S138  R=02340404 was issued.
```

From entry point ntv_advertiseWolaServer at compile unit offset +0000000020DF12E6 at entry offset +0000000000039D76 at address 0000000020DF12E6.

The messages.log file has very little other information about this error, other than the *lack* of the following message:

CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with the Liberty profile server using the following name: *<three-part name>*

The corrective action is to use a three-part name that is *unique* on the LPAR. Unfortunately, there is no easy way to check for what three-part names are currently in use. You have to know what values are coded in the server.xml files that are part of started Liberty instances.

Sample JCL

This section contains sample JCL to perform z/OS Connect EE functions.

Creating a server

The JCL below is an example of how a z/OS Connect EE server can be created using JCL.

```
//ZCEESRVR JOB (0),'ZCEE DEPLOY',CLASS=A,REGION=0M,
//             MSGCLASS=H,NOTIFY=&SYSUID,USER=liberty.id
//*************************************************************
//*  Use the zosconnect command to create a server
//*************************************************************
// SET ZCEECMD='/shared/IBM/zosconnect/v3r0/bin/zosconnect'
// SET SERVER='testsrv1'
//ZCEECMD  EXEC PGM=BPXBATSL,REGION=0M,MEMLIMIT=4G,
//   PARM='PGM &ZCEECMD. create &SERVER --template=zosconnect:default'
//STDOUT   DD   SYSOUT=*
//STDERR   DD   SYSOUT=*
//STDIN    DD   DUMMY
//STDENV   DD   *
JAVA_HOME=/shared/java/J8.0_64
WLP_USER_DIR=/var/ats/zosconnect
```

Deploying an API AAR file

The JCL below is an example of how an API AAR can be deployed using JCL.

```
//*************************************************************
//*  Step APIDPLOY - Use the apideploy commands to deploy an API
//*************************************************************
//APIDPLOY EXEC PGM=IKJEFT01,REGION=0M
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDENV   DD *
ZCEEPATH=/usr/lpp/IBM/zosconnect/v3r0
PATH=/usr/lpp/java/J8.0_64/bin:$PATH
JAVA_HOME=/usr/lpp/java/J8.0_64
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 BPXBATCH SH $ZCEEPATH/bin/apideploy -deploy +
 -a /u/johnson/Filea.aar +
 -p /var/zosconnect/servers/myServer/resources/zosconnect/apis +
    1> /tmp/zceeStd.out 2> /tmp/zceeStd.err
//*************************************************************
//*  Step COPY - Copy the apideploy commmand output to the job log
//*************************************************************
//COPY   EXEC PGM=IKJEFT01,DYNAMNBR=300
//SYSTSPRT DD SYSOUT=*
//ZCEEOUT  DD PATH='/tmp/zceeStd.out',PATHDISP=(DELETE,DELETE)
//ZCEEERR  DD PATH='/tmp/zceeStd.err',PATHDISP=(DELETE,DELETE)
//STDOUT   DD SYSOUT=*,DCB=(LRECL=1000,RECFM=V)
//STDERR   DD SYSOUT=*,DCB=(LRECL=1000,RECFM=V)
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD *
 OCOPY INDD(ZCEEERR) OUTDD(STDERR)
 OCOPY INDD(ZCEEOUT) OUTDD(STDOUT)
```

Copy WOLA executables to a load library

The JCL below is an example of how to copy the WOLA executables from WebSphere Liberty directory to an MVS PDSE.

```
//*********************************************************************/
//*  Step ALLOC    - Use the TSO ALLOCATE command to allocate a      */
//*                  PDSE load library                                */
//*********************************************************************/
//ALLOC EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DELETE USER1.WOLA1803.LOADLIB
  SET MAXCC=0
  ALLOC DSNAME('USER1.WOLA1803.LOADLIB') -
     NEW CATALOG SPACE(2,1) DSORG(PO) CYLINDERS -
     RECFM(U) DSNTYPE(LIBRARY)
//*********************************************************************/
//*  Step WOLACOPY - Use the cp shell command to copy the WOLA       */
//*                  executables to an MVS PDSE data set              */
//*  WLPPATH - denotes the path locating the WLP subdirectory        */
//*  DSNAME  - denotes the target PDSE data set                      */
//*********************************************************************/
//WOLACOPY EXEC PGM=IKJEFT01,REGION=0M
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDENV   DD *
WLPPATH=/usr/lpp/IBM/zosconnect/v3r0
DSNAME=USER1.WOLA1803.LOADLIB
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 BPXBATCH SH cp -Xv $WLPPATH/wlp/clients/zos/* "//'$DSNAME'" +
   1> /tmp/wolaStd.out 2> /tmp/wolaStd.err
//*********************************************************************/
//*  Step COPY - Copy the cp commmand output to the job log          */
//*********************************************************************/
//COPY   EXEC PGM=IKJEFT01,DYNAMNBR=300
//SYSTSPRT DD SYSOUT=*
//WOLAOUT  DD PATH='/tmp/wolaStd.out',PATHDISP=(DELETE,DELETE)
//WOLAERR  DD PATH='/tmp/wolaStd.err',PATHDISP=(DELETE,DELETE)
//STDOUT   DD SYSOUT=*,DCB=(LRECL=1000,RECFM=V)
//STDERR   DD SYSOUT=*,DCB=(LRECL=1000,RECFM=V)
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD *
OCOPY INDD(WOLAERR) OUTDD(STDERR)
OCOPY INDD(WOLAOUT) OUTDD(STDOUT)
```

Base64 Encoding

An authorization token must be provided when using the Swagger UI interface to test an API when security is enabled, see *Authorization* below. The authorization token consists of encoded string based on a combination of the user identity and password.

The token is not sent in the clear, it be encoded first using a base 64 representation of the concatenation of the user identity, a colon and the password. For example the encoded representation of string *Fred:fredpwd* is *RnJlZDpmcmVkcHd*. There are several ways to perform this encoding. The URL https://www.base64encode.org/ provides an internet tool for encoding authorization tokens.

If using an internet tool is not an option then the sample Java program below can be used to do then encoding locally. To use this program download an Eclipse package and add the sample Java code below to a Java project and run this Java application to do the encoding locally.

```
package com.ibm.ats.encode;
import org.apache.commons.codec.binary.Base64;
public class EncodeDecode {

public static void main(String[] args) {
// encode data on your side using BASE64
String str = "Fred:fredpwd";
byte[]   bytesEncoded = Base64.encodeBase64(str .getBytes());

System.out.println("ecncoded value is " + new String(bytesEncoded ));

// Decode data on other side, by processing encoded data
byte[] valueDecoded= Base64.decodeBase64(bytesEncoded );
System.out.println("decoded value is " + new String(valueDecoded));
    }
```

Note the imported project *org.apache.commons.codec.binary.Base64* can be found in Eclipse JAR file *commons-codec-1.4.jar* (or its equivalent based on the Eclipse package in use).

Controlling dynamic updates

Various components in the server configuration can be configured so updates, additions, or deletions of the underlying components are applied at a specified time internal or upon explicit request.

This is controlled by configuration attribute *updateTrigger* which is valid for the configuration elements shown below.

This attribute can be set to *polled* with means the server will scan and apply changes at an explicit interval. Note that this setting can increase CPU utilization and file I/O because the server will constantly be scanning the file systems looking for changes.

Another option for this attribute is *mbean*. This setting will cause the server to apply updates when initiated by an external request. For z/OS Connect this usually means an MVS modify command but it also can mean by a client using the JMX interface, for more information on the latter see URL https://www.ibm.com/support/knowledgecenter/en/SS4SVW_3.0.0/configuring/mbean_trigger.html

When a server is created using one of the provided templates, a subset of the configuration elements shown below are included in the *server.xml* file automatically. For the configuration elements other than *zosconnect_zosConnectDataXform*, the default value for *updateTrigge r* is *disabled.* The configuration element *zosconnect_zosConnectDataXform* has a default value of *updateTrigger* of *polled* with a default *pollingRate* of *2s*. This is something you may want to change. The key is to be aware of this behavior and understand the implication of polling.

```
   <!-- applicationMonitor is not applicable for z/OS Connect EE servers -->
<applicationMonitor updateTrigger="disabled" dropinsEnabled="false"/>

<!-- config requires updateTrigger="mbean" for REFRESH command support -->
 <config updateTrigger="mbean" monitorInterval="500ms"/>

<!-- zosConnect APIs -->
 <zosconnect_zosConnectAPIs pollingRate="5s" updateTrigger="disabled"    />

<!-- zosConnect API requesters -->
 <zosconnect_apiRequesters updateTrigger="disabled" pollingRate="5s"/>

<!-- zosConnect Services -->
 <zosconnect_services pollingRate="5s" updateTrigger="disabled"/>

<!-- zosConnect policies -->
 <zosconnect_policy pollingRate="1m" updateTrigger="disabled"/>

<!-- zosConnect data transformer -->
 <zosconnect_zosConnectDataXform pollingrate="2s" updateTrigger="polled"/>

<!—A security certificate repository -->
 <keystore pollingrate="500ms" updateTrigger="mbean"/>
```

Db2 PassTickets

z/OS Connect service level V3.0.15 added support for the use of PassTickets between a z/OS Connect server and Db2. This required some additional RACF resources which will be documented in this section.

- The PTKTDATA class was activated with a SETROPTS commands

  SETROPTS CLASSACT(PTKTDATA) RACLIST(PTKTDATA)
  SETROPTS GENERIC(PTKTDATA)

- A PTKTDATA resource was defined for the target Db2 subsystem:

  RDEFINE PTKTDATA DSN2APPL SIGNON(KEYMASK(123456789ABCDEF0)
  APPLDATA('NO REPLAY PROTECTION')

Tech-Tip: The value DSN2APPL was derived from the Db2 LU name in the DSNL004I startup message, for example.

```
DSNL004I  -DSN2 DDF START COMPLETE  906
    LOCATION  DSN2LOC
     LU       USIBMWZ.DSN2APPL
       GENERICLU -NONE
       DOMAIN    WG31.WASHINGTON.IBM.COM
       TCPPORT   2446
       SECPORT   2445
       RESPORT   2447
       IPNAME    -NONE
       OPTIONS:
       PKGREL = COMMIT
```

The value for the key mask was an arbitrary 16 hexadecimal string. If multiple RACF databases are involved this value must be the same for all.

- The identity under which the z/OS Connect server is running was given authorization to generate pass tickets for this specific PTKTDATA resource:

  RDEFINE PTKTDATA IRRPTAUTH.DSN2APPL.* UACC(NONE)
  PERMIT IRRPTAUTH.DSN2APPL.* ID(libertyID) CLASS(PTKTDATA) ACC(UPDATE)

- The RACF in storage profile need were updated:

  SETROPTS RACLIST(PTKTDATA) REFRESH)

- The server's xml *zosconnect_zosConnectServiceRestClientBasicAuth* for the connection to the Db2 subsystem was updated to replace the *userName* and *password* attributes with an applName attribute.

```
<zosconnect_zosConnectServiceRestClientConnection id="db2Conn"
      host="wg31.washington.ibm.com"
      port="2446"
      basicAuthRef="dsn2Auth" />

<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
      applName="DSN2APPL"/>
```

Db2 REST services security

This section covers a few topics related to Db2 REST services security.

- SAF class DSNR

  Access to Db2 REST services requires READ access to the Db2 subsystem DSNR resource. If a request for Db2 REST services fails to Db2 subsystem DSN2 with this message:

```
ICH408I USER(USER2   ) GROUP(SYS1    ) NAME(WORKSHOP USER2
  DSN2.REST CL(DSNR    )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
DSNL030I  -DSN2 DSNLJTIN.S30 DDF PROCESSING FAILURE
```

  Simply permit READ access to this resource to the identity in question, e.g.

```
PERMIT DSN2.REST CLASS(DSNR) ID(USER2) ACC(READ)
SETROPTS RACLIST(DSNR) REFRESH
```

- Db2 package access

  If a user is not able to display a valid Db2 REST services in the z/OS Connect Db2 services development tooling or by using a **POST** to the Db2 provided REST interface URL of http://wg31.washington.ibm.com:2446/services/DB2ServiceDiscover, then they may not have sufficient access to the package containing the service.

  For example, if service zCEEService.selectEmployee is defined to Db2 but not visible in the z/OS Connect tooling or if a **GET** request to URL

  http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee fails with message:

```
{
    "StatusCode": 500,
    "StatusDescription": "Service zCEEService.selectEmployee discovery failed due to
SQLCODE=-551 SQLSTATE=42501, USER2 DOES NOT HAVE THE PRIVILEGE TO PERFORM OPERATION
EXECUTE PACKAGE ON OBJECT zCEEService.selectEmployee. Error Location:DSNLJACC:35"
}
```

The user needs to be granted execute authority on package *zCEEService.selectEmployee* with command:

`GRANT EXECUTE ON PACKAGE "zCEEService"."selectEmployee"` or

`GRANT EXECUTE ON PACKAGE "zCEEService"."*"`

Using SAF for registry and access role checking

Up to this point Liberty has been configured to use "basic" security – that is, all security information for identities, passwords, and role access are defined in *server.xml* and managed by the Liberty server. In this section the steps required to enable authentication to a system authorization facility (SAF), e.g. RACF will be shown. For more details on this topic, see the exercise ***zCEE Customization Basic Security*** which can be found at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop.

 First, defined some basic SAF resources, e.g. RACF APPL resources.

```
ADDGROUP WSGUESTG OMVS(AUTOGID) OWNER(SYS1) 1
ADDGROUP ZCEEUSRS OMVS(AUTOGID) OWNER(SYS1) 2
ADDUSER WSGUEST RESTRICTED DFLTGRP(WSGUESTG) OMVS(AUTOUID -
HOME(/u/wsguest) PROGRAM(/bin/sh)) NAME('UNAUTHENTICATED USER') -
NOPASSWORD NOOIDCARD

ADDUSER FRED DFLTGRP(ZCEEUSRS) OMVS(AUTOUID HOME(/u/fred/) -    3
PROGRAM(/bin/sh)) NAME('USER FRED')

RDEFINE APPL BBGZDFLT UACC(NONE) OWNER(SYS1)4
PERMIT BBGZDFLT CLASS(APPL) RESET
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(WSGUEST,ZCEEUSRS)5

SETROPTS RACLIST(APPL) REFRESH  6
```

Notes:

1. Add an identity that will be used for SAF checks during the unauthenticated state prior to the actual authentication of SAF identity and password.
2. Add a group containing the authorized users of this server.
3. An example of the commands for adding a RACF identity, note that the OMVS segment with a UID is required for the identity (as well as an GID for the groups to which the user is connected).
4. Define the security prefix to be used for this Liberty server.
5. Permit the unauthenticated identity and other groups to have access to this APPL resource.
6. Permit the members of group LIBGRP access to this APPL resource.
7. Refresh the in storage for the APPL resources.

Tech Tip: The value *BBGZDFLT* in the above commands must match the value of attribute *profileprefix* in the *saf.xml* file described on the next page.

 Next, defined the required EJBROLE resource and grant access, see below.

```
RDEFINE EJBROLE BBGZDFLT.zos.connect.access.roles.zosConnectAccess -  1
OWNER(SYS1) UACC(NONE)
PE BBGZDFLT.zos.connect.access.roles.zosConnectAccess CLASS(EJBROLE) RESET
PE BBGZDFLT.zos.connect.access.roles.zosConnectAccess -    2
  CLASS(EJBROLE) ID(ZCEEUSRS) ACCESS(READ)
SETR RACLIST(EJBROLE) REFRESH                    3
```

Notes:

1. Defines the EJBRole required by z/OS Connect, e.g. *zos.connect.access.roles.zosConnectAccess*. using the value defined in the APPL resources, e.g. *BBGZDFLT,* as the resource's prefix.
2. Permit authorized users to this EJBRole resource.
3. Refresh the in storage EJBrole profiles.

 The *server.xml* needs to be changed to remove the current 'basic' configuration elements and replace them with the elements for enabling SAF security. Basic security was enabled by including *basic.xml* file in the main *server.xml* file (see *Setup of basic security* on page 26). SAF security can be enabled by creating an *saf.xml* file and replacing the include *basic.xml* to an include of *saf.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="saf security">

    <!-- Enable features -->
    <featureManager>
        <feature>appSecurity-2.0</feature>
        <feature>zosSecurity-1.0</feature>    1
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>  2

    <webAppSecurity allowFailOverToBasicAuth="true" />

    <safRegistry id="saf" />                        3
    <safAuthorization racRouteLog="ASIS" />
    <safCredentials unauthenticatedUser="WSGUEST"
       profilePrefix="BBGZDFLT" />       4
```

Notes

1. The *zosSecurity-1.0* feature adds the z/OS security feature
2. This not-SAF trust store will still be required until a SAF key ring is configured.
3. The *safRegistry, safAuthorization* and *safCredentials* elements enable authentication and authorization using SAF.
4. The profilePrefix attribute must match value of the APPL resource

 Refresh the z/OS Connect server configuration with MVS command

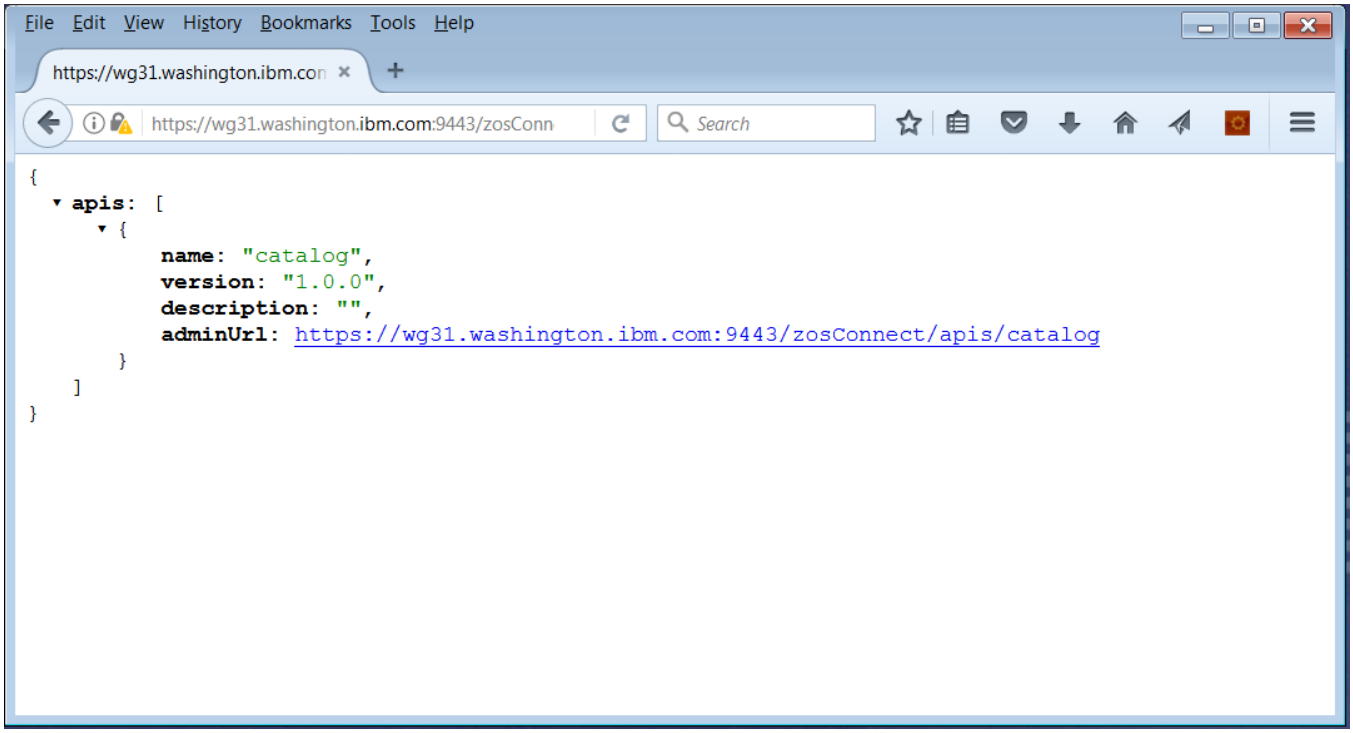
F BAQSTRT,ZCON,REFRESH

 Close all instances of the Firefox browser (we want to force another prompt for ID, and closing the browser clears any authorization tokens from the browser's cache).

 Start Firefox and enter the following URL: ***https://wg31.washington.ibm.com:9443/zosConnect/apis***

 In the userid/password prompt, enter ***Fred*** and ***FRED*** (the SAF identity and password from above).

 You should see a list of the APIs:



 Close the browser again and restart it and access the same URL. This time enter another identity,e.g, USER2, not permitted to the EJBRole.

 The request should fail with message *Error 403: AuthorizationFailed.* Check the system log using SDSF if using RACF you should an ICH408I message (see below). USER2 does not have access to the EJBROLE resource protecting the z/OS Connect server.

```
ICH408I USER(USER2   ) GROUP(SYS1    ) NAME(WORKSHOP USER2
   BBGZDFLT.zos.connect.access.roles.zosConnectAccess
   CL(EJBROLE )
   INSUFFICIENT ACCESS AUTHORITY
   ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

Summary

The registry and authorization information was removed from the *server.xml*, and other XML elements were to configure using SAF as security registry (for userid and password) and role checking (EJBROLE).

.

Using SAF for controlling z/OS Connect EE access

Currently there are no restrictions on what actions an authenticated user can performed. In this section the steps required to control SAF authorization of administrative and API execution functions will be shown. Identity FRED will have administrative authority and USER1 will only have API execution authority. In this section the steps required to enable authentication to a system authorization facility (SAF), e.g. RACF will be shown. For more details on this topic see the exercise ***zCEE Customization Basic Security*** which can be found at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop.

 Two new groups will be added using the ***ADDGROUP*** command, e.g.

- ***ADDGROUP GMADMIN OMVS(AUTOGID)***
- ***ADDGROUP GMINVOKE OMVS(AUTOGID)***

 Connect user FRED to group *GMADMIN* using the ***CONNECT*** command, e.g.

- ***CONNECT FRED GROUP(GMADMIN)***

 Connect user USER1 to group *GMINVOKE* using the ***CONNECT*** command, e.g.

- ***CONNECT USER1 GROUP(GMINVOKE)***

 Add the configuration elements below to the *server.xml*.

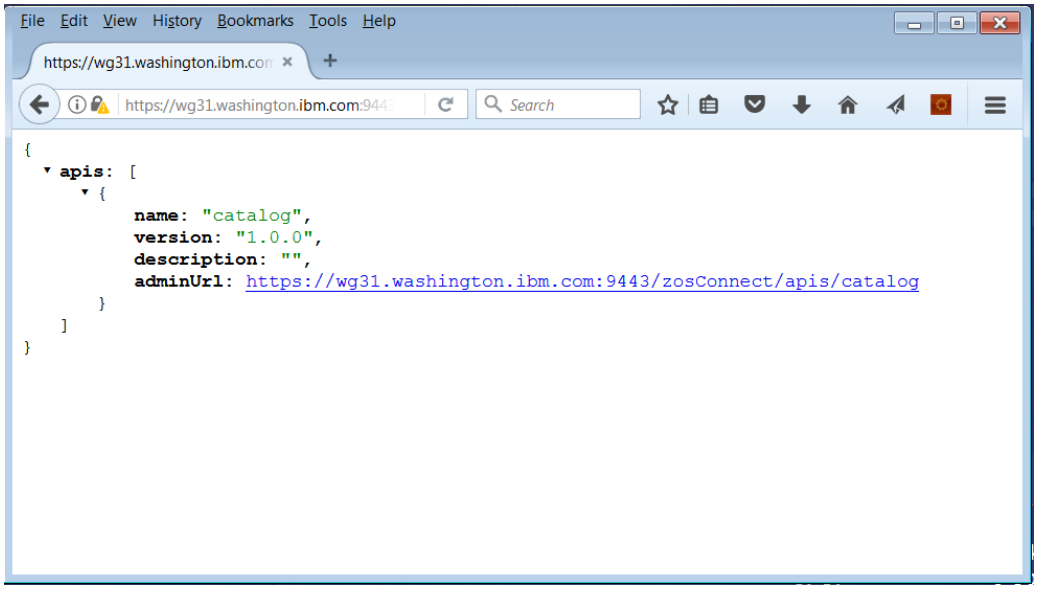
```
<zosconnect_zosConnectManager
         globalInterceptorsRef="interceptorList_g"
         globalAdminGroup="GMADMIN"
         globalInvokeGroup="GMINVOKE"/>

<zosconnect_authorizationInterceptor id="auth"/>

<zosconnect_zosConnectInterceptors id="interceptorList_g"
       interceptorRef="auth" />
```

 Stop and restart the z/OS Connect server.

 Close all instances of the Firefox browser (we want to force another prompt for ID, and closing the browser clears the security token).

 Start Firefox and enter the following URL https://wg31.washington.ibm.com:9443/zosConnect/apis.

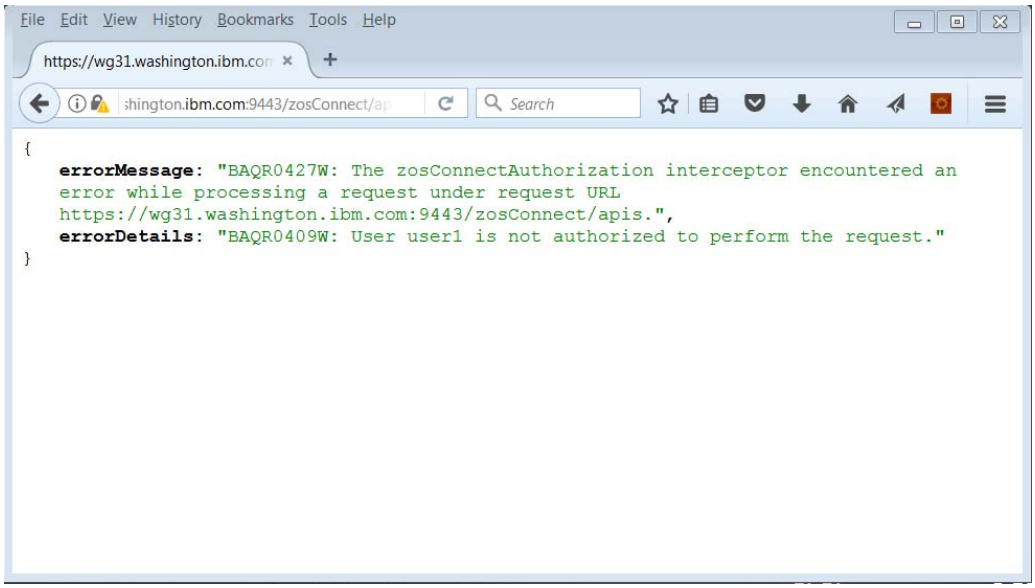
 On the *Authentication Required* popup window enter, enter ***Fred*** and ***FRED***. You should see:



FRED in in the administrators group and has the authority perform this function.

 Close Firefox session to clear the security token and restart and access the same URL.

 On the *Authentication Required* popup enter ***USER1*** and USER1's password of USER1. You should see:



Next try to invoke an API.

 Enter the command below at a command prompt and press **Enter**.

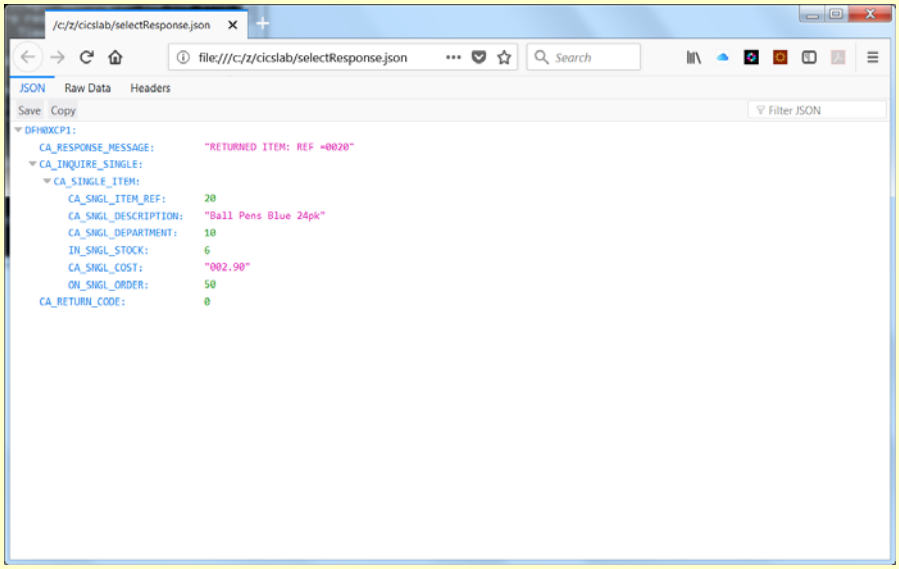
curl -X POST --user USER1:USER1 --header "Content-Type: application/json"
-d @inquireSingle.json --insecure
https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?action=invoke

 You should see the response below:

```
{"DFH0XCP1":{"CA_RESPONSE_MESSAGE":"RETURNED ITEM: REF =0020","CA_INQUIRE_SINGLE
":{"CA_SINGLE_ITEM":{"CA_SNGL_ITEM_REF":20,"CA_SNGL_DESCRIPTION":"Ball Pens Blue
 24pk","CA_SNGL_DEPARTMENT":10,"IN_SNGL_STOCK":6,"CA_SNGL_COST":"002.90","ON_SNG
L_ORDER":50}},"CA_RETURN_CODE":0}}
```

USER1 can invoke the service but has no administrative authority.

Tech Tip: Adding the *-o* flag to the cURL command will write the JSON response message to a file rather than back to the terminal session. So if you add ***-o selectResponse.json*** to the cURL command and use the command ***firefox file:///c:\z\cicslab\selectResponse.json*** you will see a browser session open with the JSON response formatted as below:



 To demonstrate an operational function, paste the command below at the command prompt and press **Enter**.

curl -X PUT --user USER1:USER1 --insecure https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?status=stopped

 You should see the response below:

```
{"errorMessage":"BAQR0406W: The zosConnectAuthorization interceptor encountered
an error while processing a request for service under request URL https://wg31.
washington.ibm.com:9443/zosConnect/services/inquireSingle.","errorDetails":"BAQR
0409W: User USER1 is not authorized to perform the request."}
```

USER1 can invoke the service but has no administrative authority.

Using RACF for TLS and trust/key store management

Authentication as configured now requires a user identity and password. Providing an identity and password is not always feasible and that case digital certificates can be used for authentication. This section shows the steps required to add support for digital certificates to the z/OS Connect server (Liberty). In this section the steps required to enable authentication to a system authorization facility (SAF), e.g. RACF will be shown. For more details on this topic, see the execise ***zCEE Customization Basic Security*** which can be found at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop.

 First, defined some basic SAF resources, e.g. RACF digital certificates.

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty') -    1
  OU('LIBERTY')) WITHLABEL('Liberty CA') TRUST -
  SIZE(2048) NOTAFTER(DATE(2022/12/31))
RACDCERT CERTAUTH EXPORT(LABEL('Liberty CA')) -               2
  DSN('USER1.CERTAUTH.CRT') FORMAT(CERTDER)
RACDCERT ID(LIBSERV) GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com') -  3
  O('IBM') OU('LIBERTY')) WITHLABEL('Liberty Client Cert') -
  SIGNWITH(CERTAUTH LABEL('Liberty CA')) SIZE(2048) -
  NOTAFTER(DATE(2022/12/31))
RACDCERT ID(LIBSERV) ADDRING(Liberty.KeyRing)                 4
RACDCERT ID(LIBSERV) CONNECT(ID(LIBSERV) -
  LABEL('Liberty Client Cert') RING(Liberty.KeyRing)DEFAULT)  5
RACDCERT ID(LIBSERV) CONNECT(CERTAUTH LABEL('Liberty CA') -    6
  RING(Libery.KeyRing))
PERMIT IRR.DIGTCERT.LISTRING -                                7
  CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST -
  CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)                    8
SETR RACLIST(FACILITY) REFRESH                                9
```

Notes:

1. Generate a Liberty certificate authority (CA) certificate. This certificate will be used to sign and authenticate personal certificates.

2. The just create CA certificate will be exported from RACF and imported into trust stores for use by clients on other platforms. This will allow the authentication of any personal certificate signed by the CA certificate when presented to the client on the other platforms.
3. Generate a personal certificate signed by the Liberty CA certificate. This will be the personal certificate provided by the Liberty server when it needs to provide a digital certificate during a TLS handshake.
4. Create a RACF key ring for managing certificates. This key ring will below to the RACF identity under which the z/OS Connect is running.
5. Connect or attach the z/OS Connect personal certificate to the z/OS Connect server's key ring.
6. Connect or attach the Liberty CA certificate to the z/OS Connect server's key ring.
7. Permit the z/OS Connect server access to its own key ring.
8. Permit the z/OS Connect server access to its own certificate.
9. Refresh the FACILITY class in storage profiles.

___1. Next, create and export additional personal certificates for use in authenticating other users.

```
RACDCERT ID(FRED) GENCERT SUBJECTSDN(CN('Fred D. Client') -   1
  O('IBM') OU('LIBERTY')) WITHLABEL('FRED') -
  SIGNWITH(CERTAUTH LABEL('Liberty CA')) SIZE(2048) -
  NOTAFTER(DATE(2022/12/31))
RACDCERT ID(FRED) EXPORT(LABEL('FRED')) -             2
  DSN('USER1.FRED.P12') FORMAT(PKCS12DER) -
  PASSWORD('secret')
RACDCERT ID(FRED) EXPORT(LABEL('FRED')) -          3
  DSN('USER1.FRED.PEM') -
  PASSWORD('secret')
RACDCERT ID(USER1) GENCERT SUBJECTSDN(CN('USER1 D. Client') -   4
  O('IBM') OU('LIBERTY')) WITHLABEL('USER1') -
  SIGNWITH(CERTAUTH LABEL('Liberty CA')) SIZE(2048) -
  NOTAFTER(DATE(2022/12/31))
RACDCERT ID(USER1) EXPORT(LABEL('USER1')) -     5
  DSN('USER1.USER1.P12') FORMAT(PKCS12DER) -
  PASSWORD('secret')
RACDCERT ID(USER1) EXPORT(LABEL('USER1')) -    6
  DSN('USER1.USER1.PEM') -
  PASSWORD('secret')
SETR RACLIST(DIGTCERT DIGTRING) REFRESH    7
```

Notes:

1. Generate a personal certificate for identity FRED signed with the Liberty CA certificate.
2. Export FRED's personal certificate encrypted and protected with a password.
3. Export FRED's personal certificate in PEM format (universal format).
4. Generate a personal certificate for identity USER1 signed with the Liberty CA certificate.
5. Export USER1's personal certificate encrypted and protected with a password.
6. Export USER2's personal certificate in PEM format (universal format).
7. Refresh the digital certificate and key ring in in storage profiles.

Tech-Tip: The personal certificates are being exported so they can be moved to other platforms. On the other platforms they will be used by various clients as means to identify themselves to the z/OS Connect server.

 Update the z/OS Connect server's *server.xml* by adding a new feature (*transportSecurity*) to the existing *featureManager* list and SSL related configuration elements, see below:

```
    <featureManager>
        <feature>transportSecurity-1.0</feature>      1
    </featureManager>

     <sslDefault sslRef="DefaultSSLSettings" />   2
     <ssl id="DefaultSSLSettings"
        keyStoreRef="CellDefaultKeyStore"
        trustStoreRef="CellDefaultKeyStore" />
     <keyStore id="CellDefaultKeyStore"          3
        location="safkeyring:///Liberty.KeyRing"
        password="password" type="JCERACFKS"
        fileBased="false" readOnly="true" />
```

Notes

1. *transportSecurity-1.0* feature enables TLS support
2. The use of DefaultSSLSettings specifies the default *ssl* configuration element.
3. The *keystore* element identity the RACF keyring containing the CA and personal certificates and replaces the previous non-SAF trust store.

Tech-Tip: The *password* attribute is required but is not used on z/OS. It still should be set to *password*. On z/OS the keyring is identified by the SAF user under which the task is executeing.

 Stop and restart the server.

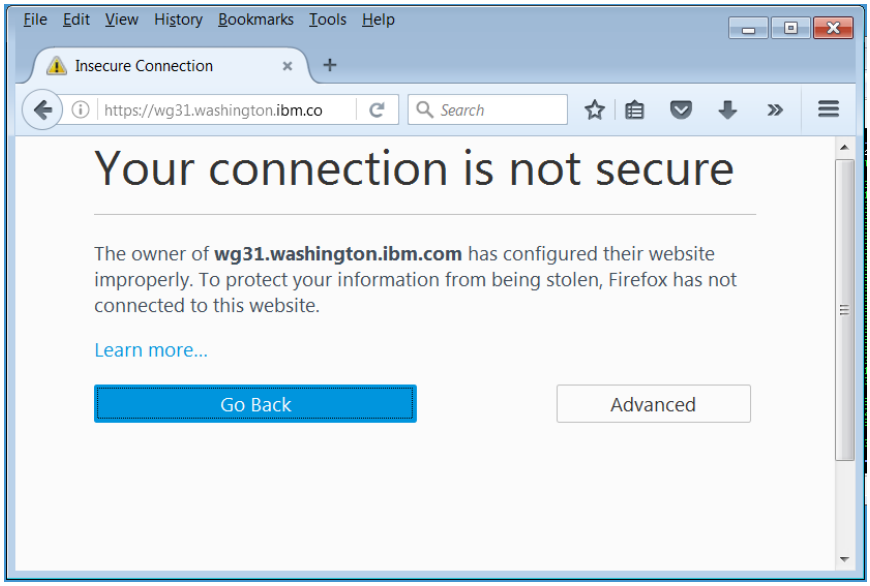
 Close all instances of your Firefox browser[18].

 Start Firefox and issue the following URL:

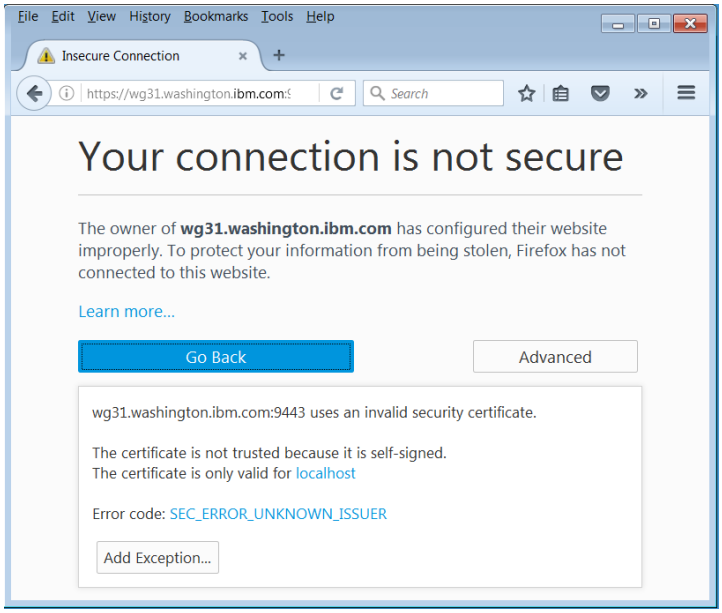
https://wg31.washington.ibm.com:9443/zosConnect/apis

18 So the certificate accepted earlier is cleared and you're forced to see the new SAF-created certificate.

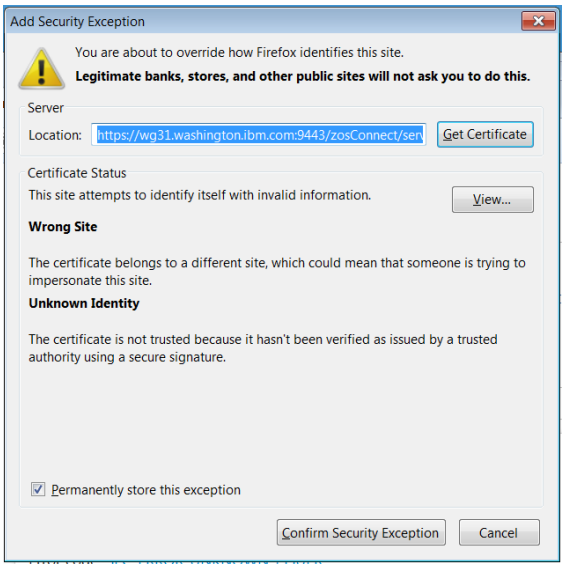
A challenged by Firefox will be display because the digital certificate used by the Liberty z/OS server does not recognize RACF signed certificates. Click on the **Advanced** button to continue.



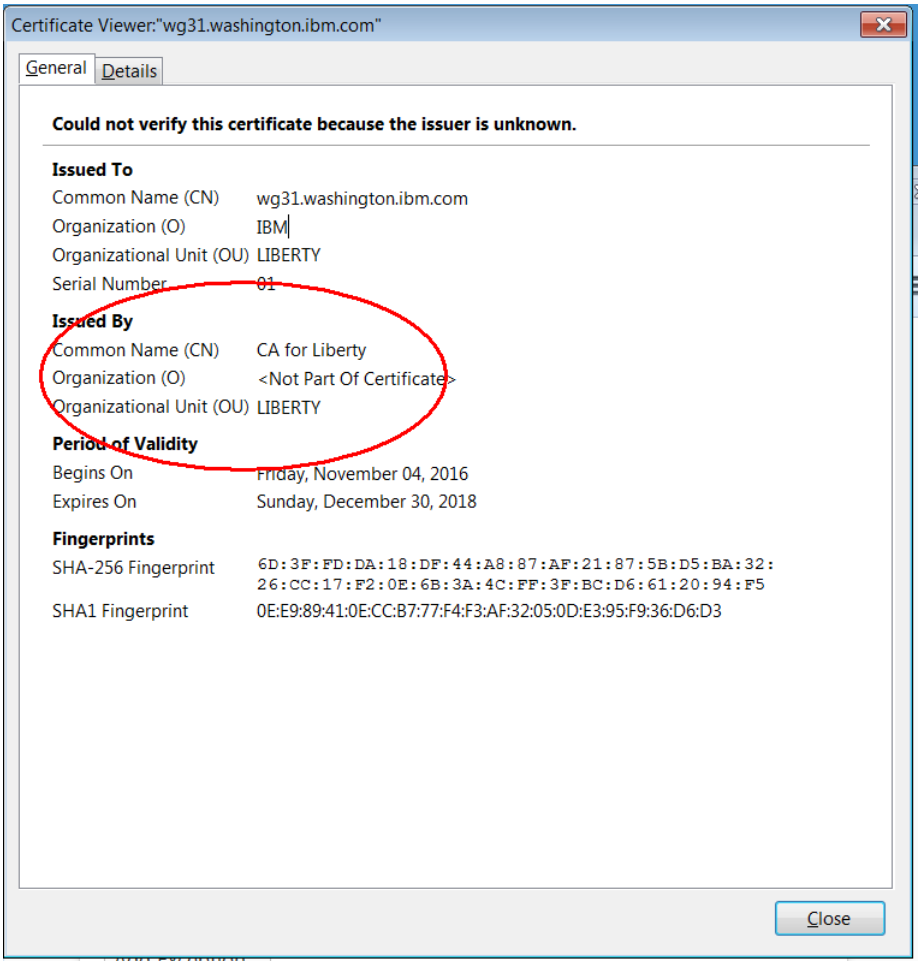
- Click the **Add Exception** button to continue.



 Click on the **View** button to display details about the certificate.



 This Certificate Authority (CA) that issued this certificate does not exist in the trust store used by Firefox. Click the **Close** button to continue.

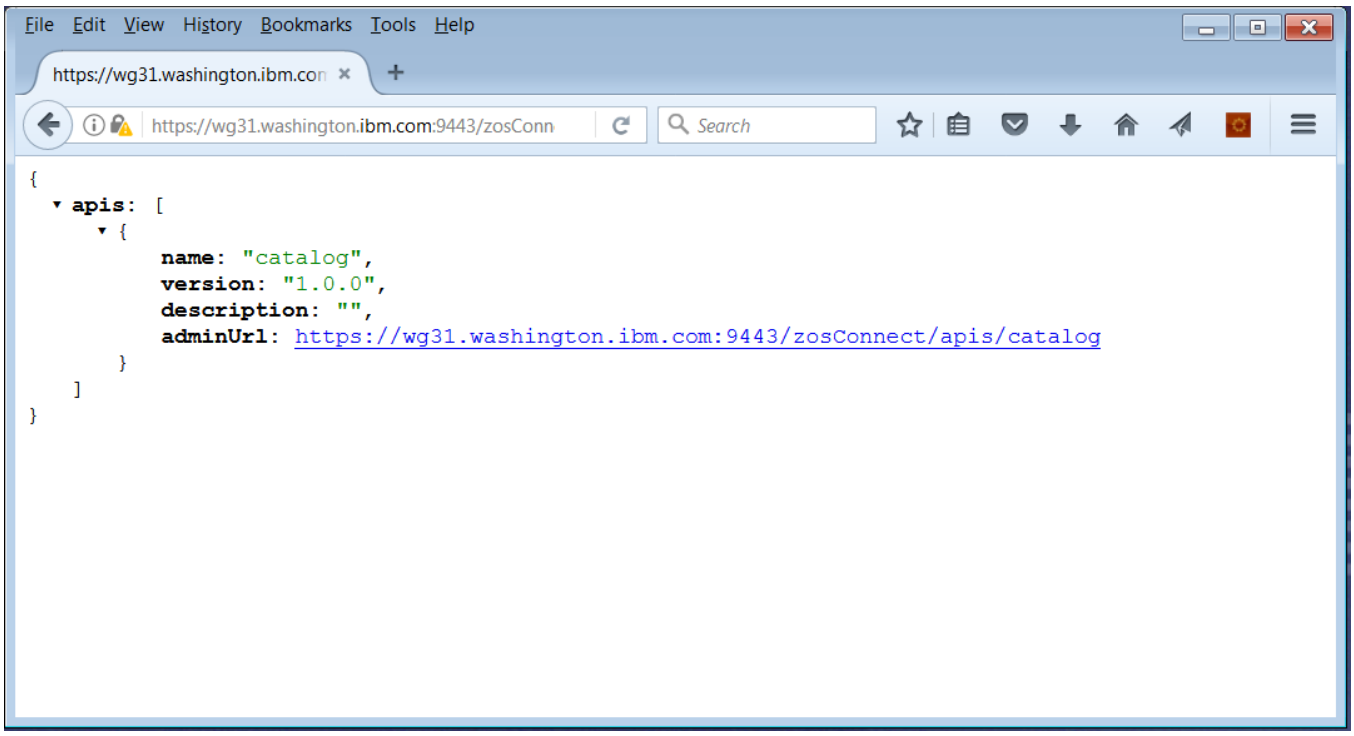


 Click on the **Confirm Security Exception** button.

 In the userid/password prompt window enter ***Fred*** and ***Fred***'s password.

With SAF case does not matter. All userid and password values are stored in upper-case. Anything entered in lowercase or mixed is folded to uppercase and compared against the SAF registry.

 You should see a familiar list of APIs:



Summary

One more element of the security infrastructure was moved from the "basic" Liberty implementation down into SAF. In this case it was the certificates for the establishment of the encrypted link.
In the "real world" a known Certificate Authority (such as VeriSign) would be used to sign the server certificate. In that case the browser would trust the certificate based on the well-known CA and you would not get a challenge.

Using client certificates for authentication

Up until now the server has been sending its personal certificate for the client to validate with its local copy of the CA certificate in its trust store. It is also possible to have the client send its personal certificate to the z/OS Connect for validation with the CA certficate connected to the server key ring. Once this client certificate has been validated the SAF identity associated with that certificate can be used for subsequent authorization checks. This section describes the steps to implement this exchange of certificates between the client and server which is also known as mutual authentication. In this section the steps required to enable authentication to a system authorization facility (SAF), e.g. RACF will be shown. For more details on this topic, see the exercise ***zCEE Customization Basic Security*** which can be found at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop.

 Stop the the z/OS Connect server.

 Update the default configuration element by adding the lines in bold below:

```
<sslDefault sslRef="DefaultSSLSettings" />
<ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultTrustStore"
    clientAuthenticationSupport="true"      1
    clientAuthentication="true" />          2
```

Notes

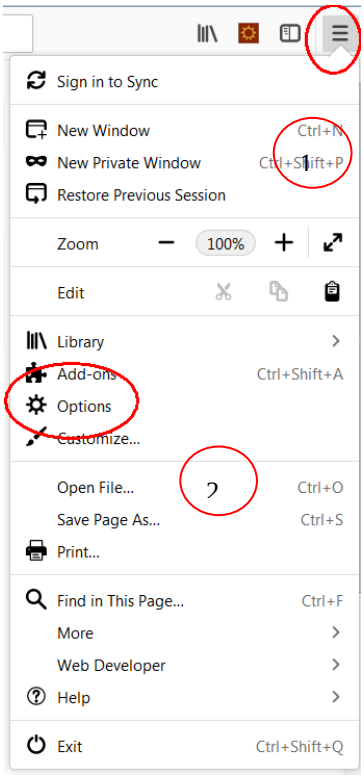
1. If set to *true* and the client presents a personal certificate it will be validated during the handshake process, e.g. mutual authentication is enabled.
2. Client authentication is required when set to *true.*

 Download the exported certificate authority and personal certificates to:

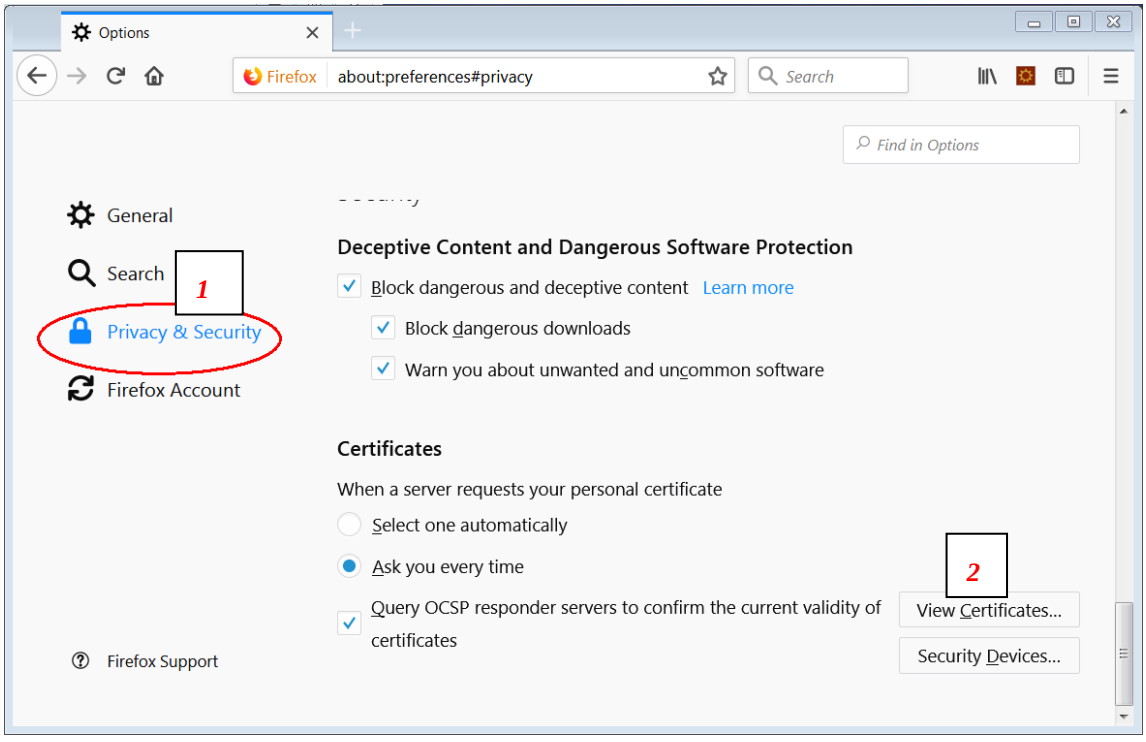
- Certificates exported in PEM format should be downloaded in ASCII mode, e.g. USER1.FRED.PEM.
- Certificates exported in PKCS12DER format should be download in Binary mode, e.g. USER1.FRED.P12.
- Certificates exported in CERDER format should be downloaded, e.g. USER1.CERTAUTH.CRT.

With the certificates downloaded, the next step is to import them into Firefox. That's next.

 In Firefox, click on the to the *Open Menu* (1) icon and select the *Options* (2) tool.



 Click on *Privacy & Security* (1) then scroll down to the *Certificates* (2) tab:

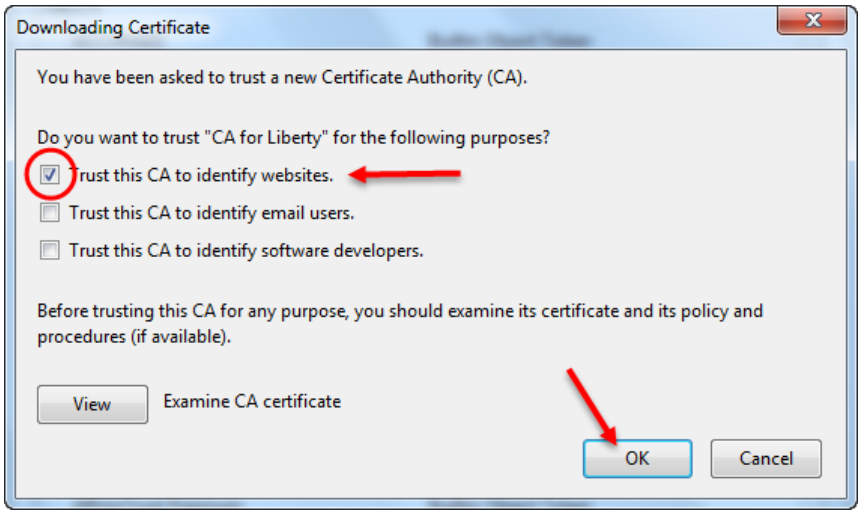


Then click the **View Certificates** button.

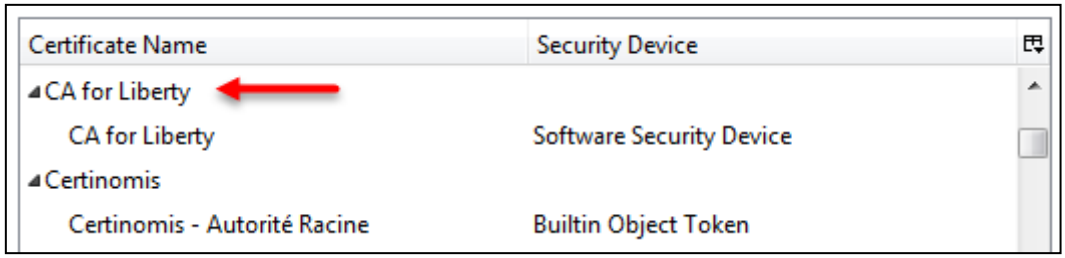
 Then click on the *Authorities* tab, and the **Import** button.

 Navigate to to the directory to where the **certauth.crt** file was downloaded and double-click on the **certauth.crt** file.

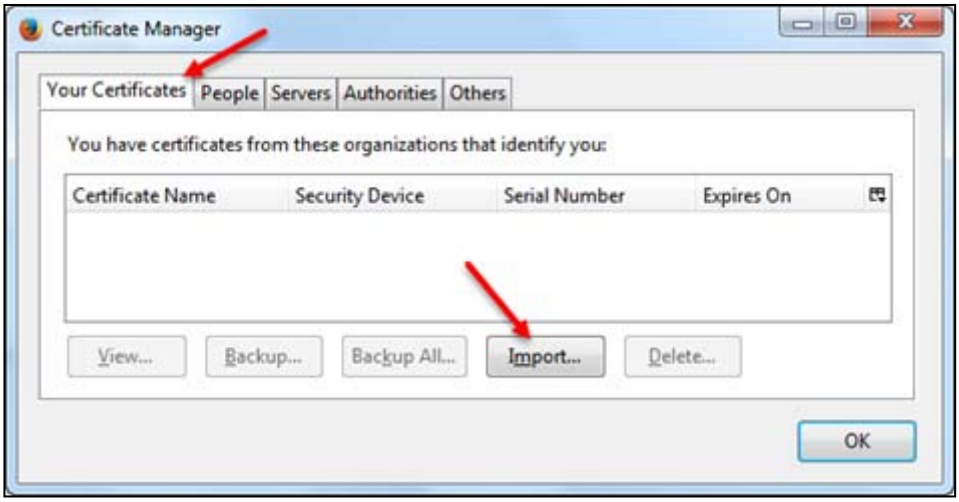
Then check the *Trust this CA to identify websites* box and click **OK:**



Verify the certificate has been imported by scrolling down and looking for the "CA for Liberty" certificate in the list:

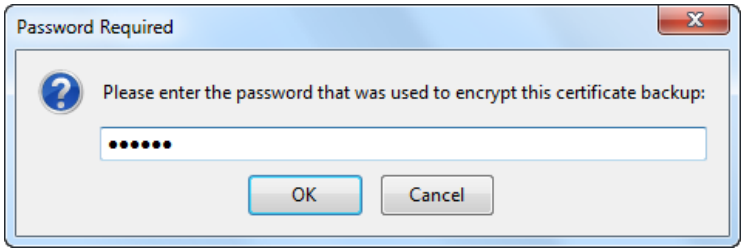


Next, click the *Your certificates tab* and then the **Import** button:



□ It should open up at the same directory from before, but if not then navigate to that location. Locate the **fred.p12** certificate and double-click on it.

A window will appear asking you to enter the password for the certificate:



Enter the value[19] ***secret*** and click **OK**. You should see confirmation:

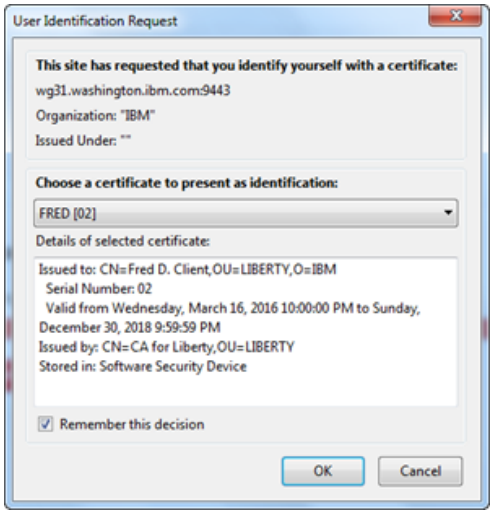
□ Click **OK** to clear the confirmation, then

□ **OK** to close the certificate manager panel, **OK** to close the options panel, and then close *all instances* of your Firefox browser.

□ Restart your server.

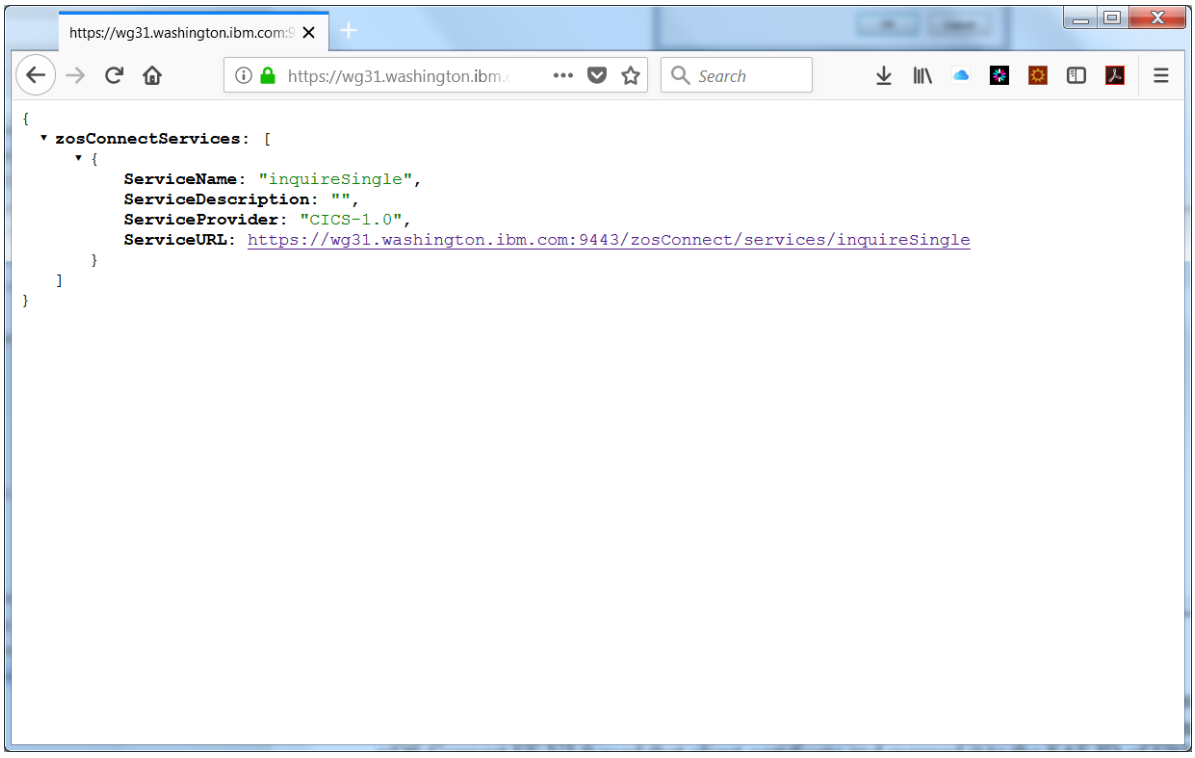
□ Start Firefox and go to URL ***https://wg31.washington.ibm.com:9443/zosConnect/services***

You will be prompted for which client certificate you wish to use:



□ You only have one, and it's selected ... so click **OK**.

□ You should see the list of installed services:



□ Enter the command below at a command prompt and press **Enter**.

curl -X put --cacert certauth.pem --cert user1.p12:secret --cert-type P12 https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?action=start

 You should see the response below:

```
{"errorMessage":"BAQR0406W: The zosConnectAuthorization interceptor encountered
an error while processing a request for service inquireSingle under request URL
https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle.","errorD
etails":"BAQR0409W: User USER1 is not authorized to perform the request."}
```

The USER1 identity is determined by the client certificate specified in user1.p12.

 Enter the command below at a command prompt and press **Enter**.

curl -X put --cacert certauth.pem --cert fred.p12:secret --cert-type P12 https://wg31.washington.ibm.com:9443/zosConnect/services/inquireSingle?action=start

 You should see the response below:

```
{"zosConnect":{"serviceName":"inquireSingle","serviceDescription":"","servicePro
vider":"CICS-1.0","serviceURL":"https://wg31.washington.ibm.com:9443/zosConnect/
services/inquireSingle","serviceInvokeURL":"https://wg31.washington.ibm.com:9443
/zosConnect/services/inquireSingle?action=invoke","dataXformProvider":"zosConnec
tWVXform-1.0","serviceStatus":"Started"}}
```

The FRED identity is determined by the client certificate specified in fred.p12 and FRED has administrator authority.

RACF Certificate Mapping and Filtering

Rather than creating or maintaining digital certificates for every user we can create a mapping that can be used to associate a RACF identity will any valid digital certificates where the subject's distinguished name and/or the issuer's distinguished name matches a pattern or filter.

 Filters can be created with a RACDCERT command. Enter command RACDCERT ID MAP to create a filter that assigns RACF identity ATSUSER to any digital certificate signed with the ATS client signer certificate and where the subject is organizational unit ATS in organization IBM.

```
racdcert id(atsuser) map sdnfilter('OU=ATS.O=IBM')  idnfilter('CN=ATS Client
CA.OU=ATS.O=IBM')withlabel('ATS USERS')
```

 Enter command RACDCERT ID MAP to create a filter that assigns RACF identity OTHUSER to any digital certificate signed by the ATS client signer certificate and where the subject is in organization IBM.

```
racdcert id(othuser) map sdnfilter('O=IBM') idnfilter('CN=ATS Client
CA.OU=ATS.O=IBM')  withlabel('IBM USERS')
```

Tech-Tip: The commands in these examples were entered in mixed case in order to emphasize the case sensitivity of the filter values and labels in these commands. The values for the common name (CN), organizational unit (OU) and organization(O) in the subject's and issuer's distinguished name filters (sdnfilter and idnfilter) must match the value and case specified in the original certificate request .Using "o=ibm" in the generate key request will not match a filter or map created with 'O=IBM' in sdnfilter.

 Enter command SETROPTS refresh the in storage profiles for the digital certificates maps.

```
setropts raclist(digtnmap) refresh
```

Now any valid client certificate presented to the z/OS Connect server issued by a CA named CN=ATS Client CA.OU=ATS.O=IBM with a subject of OU=ATS.O=IBM will use identity ATSUSER for any authorization checks. Other valid client certificated presented to the z/OS Connect server issued by the same CA but with a subject of O=IBM (OU is value other than ATS) will use OTHUSER for any subsequent authorization checks

Summary

In the web browser you were prompted for a client certificate (because of an option that defaulted when you imported the client certificate). z/OS Connect used that client certificate and mapped it to the SAF ID of FRED. That's what allowed you to invoke the *zosConnect/services* API and get the list of services.

In the cURL example the client certificate specified by the *–cert* flag determined which identity was used for authorization checking in z/OS Connect EE because *clientAuthenication* was enabled.

CICS Identity Propagation

To enable the propagation of the authenticated identity onto CICS for CICS authorization check s make the perform the following steps. Use your own values for NetwortID, APPLID. In this section the steps required to enable authentication to a system authorization facility (SAF), e.g. RACF will be shown. For more details on this topic, see the ***zCEE Customization CICS Security*** which can be found at URL https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop.

 Activate the SAF IDIDMAP class, e.g. ***SETROPTS CLASSACT(IDIDMAP)***

 Define a mapping from the distributed identity to a local SAF identity, e.g.
racmap id(fred) map userdidfilter(name('Fred')) registry(name('zosConnect')) withlabel('fred')

 Refresh the IDIDMAP in store profiles, e.g. ***setropts raclist(ididmap) refresh.***

 Add *zosConnectNetworkid* and *zosConnectApplid* elements to a *zosconnect_cicsIpicConnection* configuration element.

```
<zosconnect_cicsIpicConnection id="cscvinc"
   host="wg31.washington.ibm.com"
   zosConnectNetworkid="ZOSCONN"  1
   zosConnectApplid="ZOSCONN"    2
   port="1491"/>
```

Notes:
1. The value of zosConnectNetworkid must match the value of the *NETWORKID* of the IPCONN CICS resource
2. The value of zosConnectApplid must match the value of the *APPLID* of the IPCONN CICS resource

 Define a CICS IPCONN resources using these attributes:

```
DEFINE IPCONN(ZOSCONN) GROUP(SYSPGRP)
    APPLID(ZOSCONN)  1
    NETWORKID(ZOSCONN)          2
    TCPIPSERVICE(ZOSCONN)  3
    LINKAUTH(SECUSER)
    USERAUTH(IDENTIFY)
    IDPROP(REQUIRED)
```

Notes:
1. The value of *NETWORKID* must match the value of the *zosConnectNetworkid* of the *zosconnect_cicsIpicConnection* element.

2. The value of *APPLID* must match the value of the zosConnectApplid of the *zosconnect_cicsIpicConnection* element.
3. The value of TCPIPSERVICE must match the name of the CICS TCPIPService that defines the port that corresponds to the port configured in the *zosconnect_cicsIpicConnection* element.

 Define CICS TCPIPService specifying a URM value of NO.

 The CICS region must have security enabled *(SEC=YES)*, TCP/IP enabled *(TCPIP=YES)* and intersystem communication enabled (*ISC=YES*).

Tech Tip: There will be at least one security check performed when CICS starts the mirror transaction. The security check will be for *READ* access to either transaction code *CSMI* (the CICS default mirror transaction) or the value of the transaction code specified in service's configuration for the *Transaction ID* attribute when the *Transaction ID Usage* attribute is set to *EIB_AND_MIRROR*

A SAF check will be performed with the identity propagated from z/OS Connect. But before this check, another SAF check may be performed using a *link identity*. The *link identity* is determined as follows. For an SSL connection, e.g. *LINKAUTH(CERTUSER)*, the *link identity* will be the local SAF identity mapped to the client certificate. For a non-SSL connection, e.g. *LINKAUTH(SECUSER)*, the *link identity* will be the value provided in the *SECURITYNAME* IPCONN attribute. If no value is provided in this attribute, the CICS default user identity will be used for the *link identity*.

If the *link identity* matches the SAF identity under which the CICS region is running, only the propagated identity is used for a SAF check for access to the mirror transaction. If the *link identity* does not match the SAF identity of the CICS region then a SAF check is also performed for the *link identity's* access to the mirror transaction.

Review the CICS documentation regarding the *IDprop* attribute. Behavior of this attributes depends on whether the zCEE server and the CICS region are in the same Sysplex or not

z/OS Connect and AT-TLS

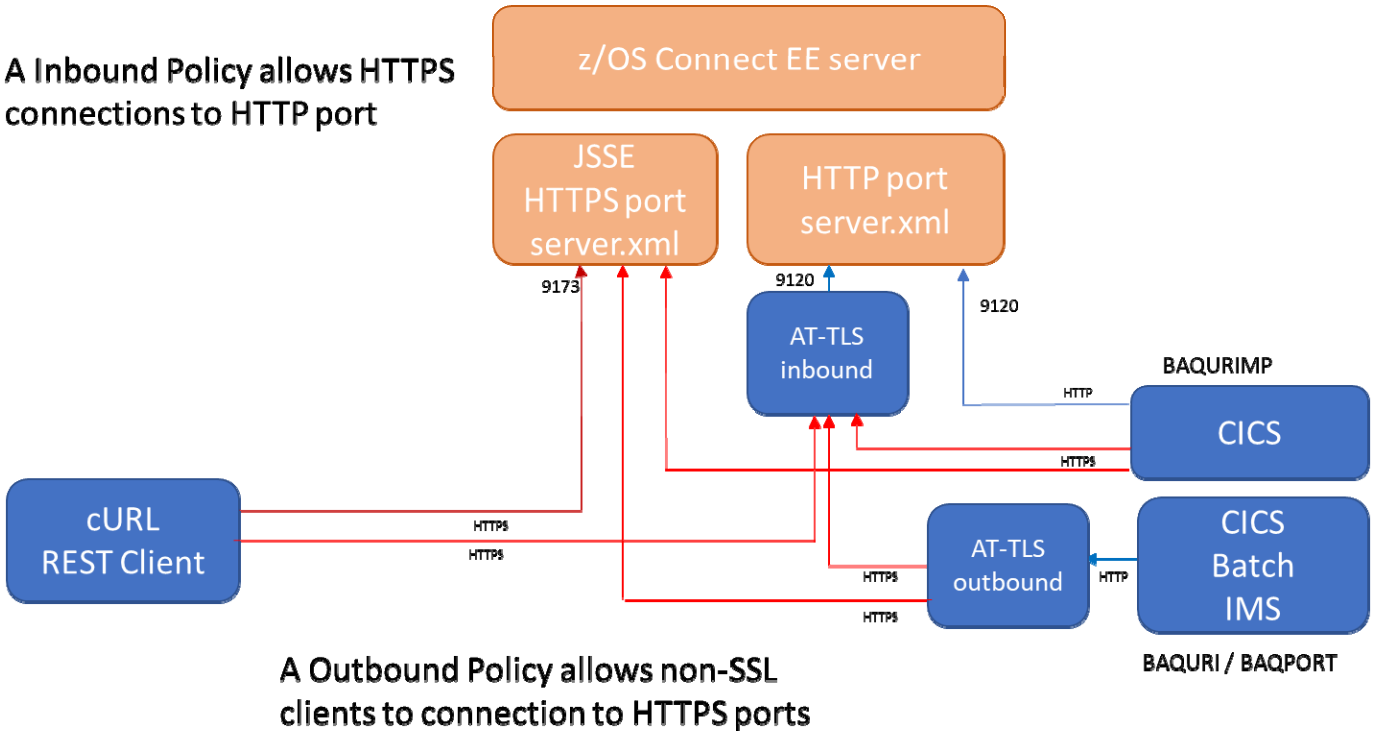
In some situations, z/OS Connect requires the presence of Application Transparent – TLS (AT-TLS) to enable encryption and security between a z/OS Connect server and an inbound REST client or an outbound service provider. AT-TLS is a component of the IBM z/OS Communication Server product (specifically the TCP IP stack) that can provide TLS support between endpoints(applications). The Transparent part of the name means that endpoints need not be aware that network traffic is being encrypted and/or digital certificates are being used for security.

AT-TLS is used when TLS is required for communications with Db2 (a Db2 requirement) and/or when TLS is required by an API client requester application running in MVS batch job or an IMS region. This document will describe configuring AT-TLS for an API requester application running in other non-CICS environments.

HTTPS Communication Options

The diagram below shows the flows for inbound communication to a z/OS Connect server. REST Clients such as Curl provide TLS support and can interact directly with the JSSE support provided by the Liberty runtime in which z/OS Connect server is running. Also, CICS SSL support provides the same functionality.

AT-TLS is required for TLS support between MVS batch and/or IMS applications to a z/OS Connect server. Two types of AT-TLS configurations or policies would be required. Inbound policy performs the functions of a TLS server for inbound HTTPS request connecting when connecting to an HTTP port and outbound policy which perform the functions of a TSL client for outbound request going from a non-TLS enabled client.



AT-TLS policies are stored in a file in an OMVS directory and contains sections for configuration for ports, traffic directions, IP addresses, key rings and ciphers, etc. All this information is not easily manageable using an editor, so the use of the *Configuration Assistant* tool provided by IBM z/OS Manager Facility (z/OSMF) is the recommended way to configure AT-TLS policies. See Redbook *IBM z/OS V2R2 Communications Server TCP/IP Implementation: Volume 4 Security and Policy-Based Networking, SG24-8363-00* for details regarding the configuring and usage of the Policy Agent and *Configuration Assistant*.

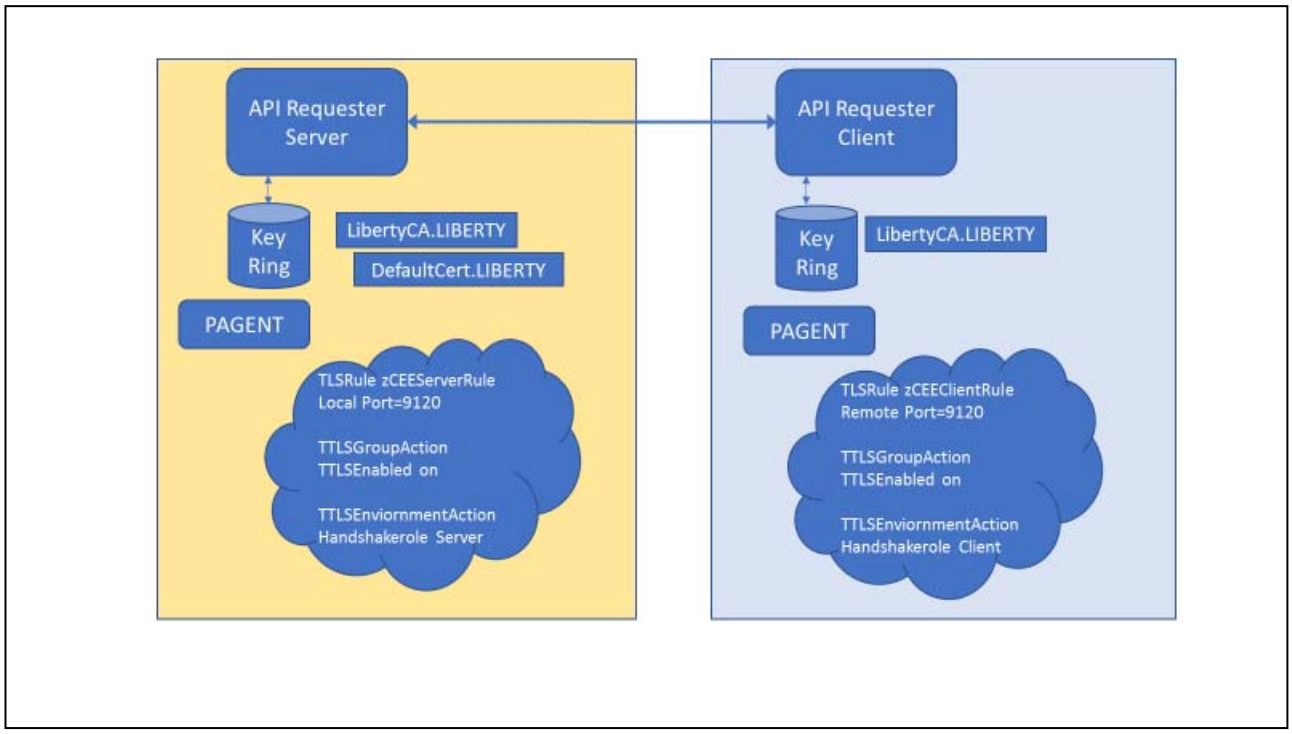
The next section shows screen shots of the significant screens shots from the *Configuration Assistant* used to configure the complete AT-TLS policy shown in *Generated AT-TLS policies* on page 113. Not all the steps for using the *Configuration Assistant* will be shown, just the key screens.

AT-TLS Configuration

Let's explore these inbound and outbound policies in a little more detail. The diagram below demonstrates both inbound and outbound policies.

The TCP/IP stack's *Policy Agent* (PAGENT) performs various functions, one of which monitors TCP/IP traffic at the transport layer and triggers AT-TLS when the properties of a network traffic request matches a set of criteria defined in an AT-TLS policy.

For the API requester server, the policy identifies the target port for an inbound request, the key ring to be used for TLS handshakes, encryptions cyphers, etc. and what role should be played by AT-TLS during a handshake, e.g. server. For the API requester client, the policy identifies the target port for an outbound request, the key ring to be used for TLS handshakes, encryptions cyphers, etc. and what role should be played by the AT-TLS during a handshake, e.g. client.



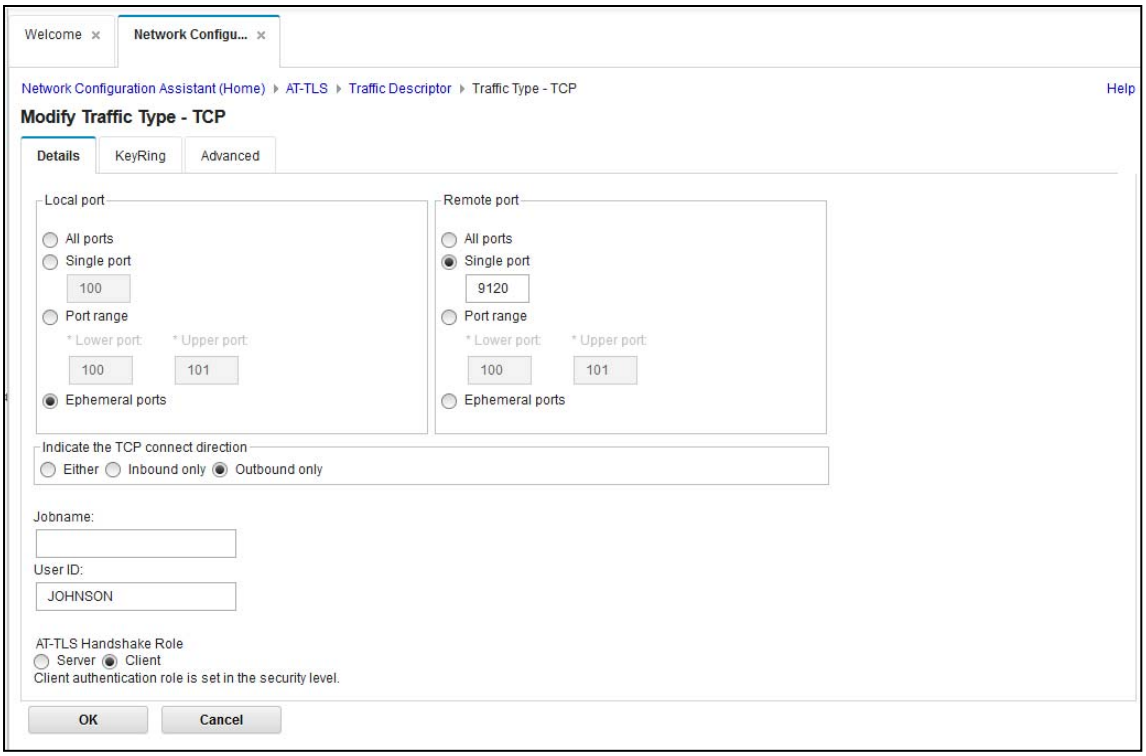
In the example above the keyring configured in the AT-TLS policy for the API requester server is the same key ring (*Keyring.LIBERTY*) created *in Using RACF for TLS and trust/key store management* on page 93. The server's end points are configured as shown below:

```
<httpEndpoint id="defaultHttpEndpoint"
              host="*" httpPort="9120" httpsPort="9173" />
```

In this example mutual authentication will not be configured so only the signer certificate of the server certificate sent by the z/OS Connect server needs to be connected to the client's key ring.

HTTP Client Traffic Descriptor

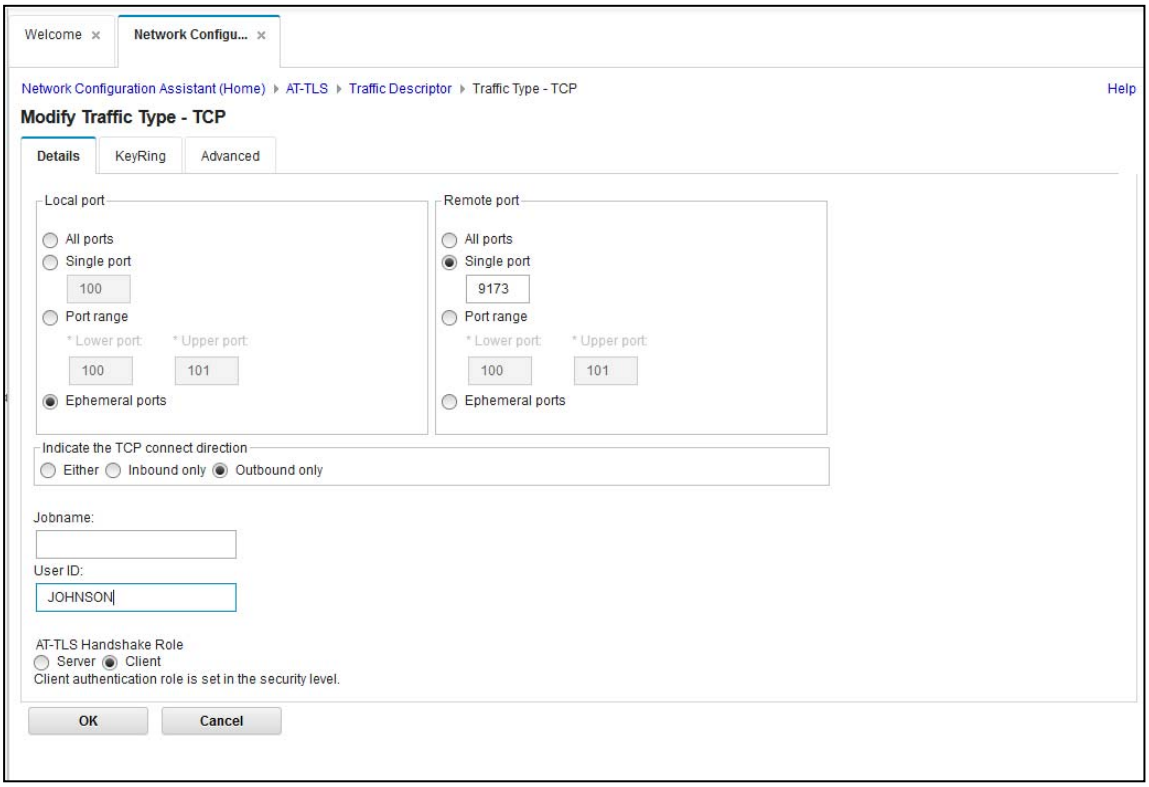
The *Configuration Assistant* identifies the target port and handshake role in a *Traffic Descriptor* component. As shown below this descriptor identifies the remote port for the server as being 9120. This descriptor applies to all inbound IP address but only if the requester is running under a SAF identity of JOHNSON. When these criteria` are met, AT-TLS will act as a client during a TLS handshake with the server. The *User ID* was provided so other clients running under other identities could connect to the server's HTTP port as normal. This policy will act as a client during a TLS hand shake. Also defined in the descriptor is the key ring, e.g. *Keyring.zCEE.*



When this traffic descriptor is combined with other definitions in a policy there will be a need to be a corresponding inbound policy to act as server during a TLS handshake (the z/OS Connect server is not involved in the TLS process at all).

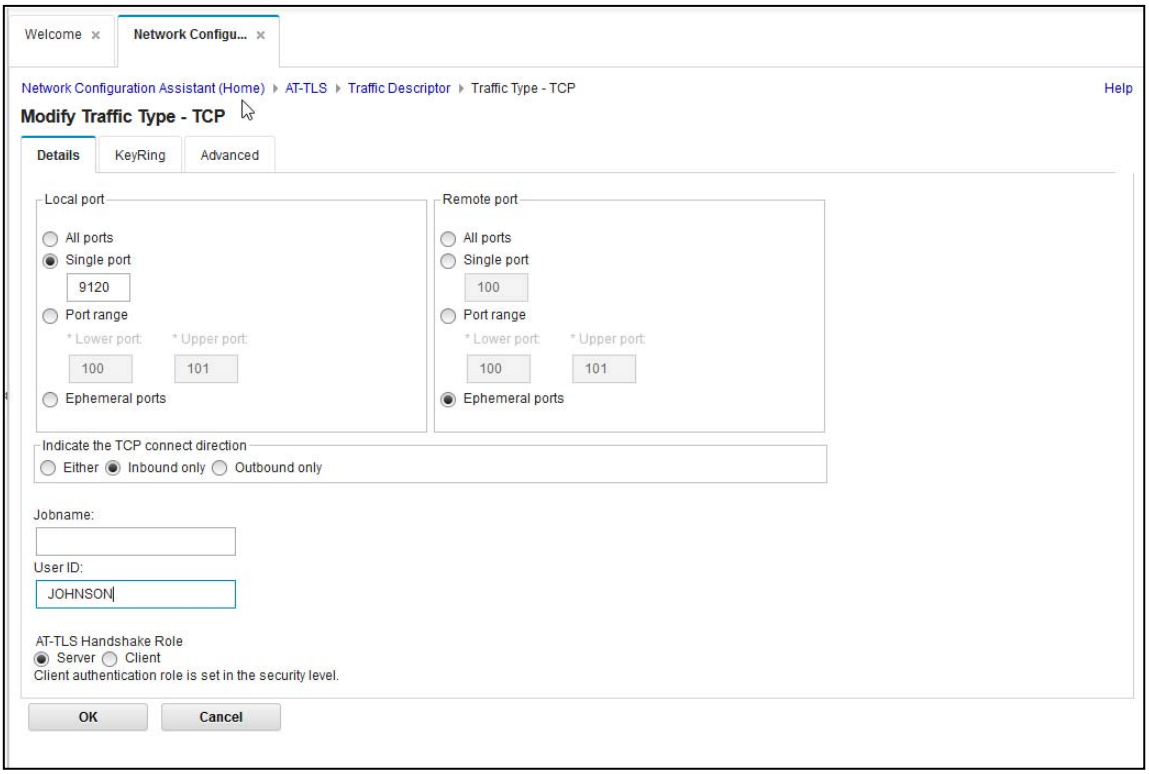
HTTPS Client Traffic Descriptor

As shown below, the traffic descriptor defines the remote port for the server as being 9173. This descriptor applies to all inbound IP address but only if the client is running under a SAF identity of JOHNSON. When these criteria are met, AT-TLS will act as a client during a TLS handshake with server. The *User ID* was provided so other clients running under other identities could connect to the server's HTTPS port as normal. This policy will act as a client during a TLS handshake. Also defined in the descriptor is the key ring, e.g. *Keyring.zCEE*



Server Traffic Descriptor

The outbound AT-TLS policy identifies the local port and handshake role in a Traffic Descriptor component. As shown below this descriptor identifies the local port for the server as being 9120. This descriptor applies to all inbound IP addresses but only if the client is running under a SAF identity of JOHNSON. The *User ID* was provided so other clients running under other identities could connect to the server's HTTP port as normal. This policy will act as a server during a TLS handshake. Also defined in the descriptor is the key ring, e.g. *Liberty.KeyRing. (This is the same key ring used by the Liberty server for JSSE handshakes shown earlier).*



When the configuration is complete in the *Configuration Assistant* it is exported to an OMVS file and the Policy Agent is told to update its configuraton with an MVS modify command,

F PAGENT,UPDATE

Note that the names of traffic discritors, rules, etc configured in the Configuration Assistance are mangled during the export process.

Generated AT-TLS policies

```
TTLSRule                                    zCEEClientRule~1
{
   LocalAddrGroupRef                     zOSConnectServers
   RemoteAddrGroupRef                     zOSConnectServers
   LocalPortRangeRef                     portR1
   RemotePortRangeRef                     portR2
   Userid                                       JOHNSON
   Direction                                 Outbound
   Priority                                    255
   TTLSGroupActionRef                     gAct1
   TTLSEnvironmentActionRef            eAct1
   TTLSConnectionActionRef            cAct1
}
TTLSRule                                    zCEEClientSSLRule~2
{
   LocalAddrGroupRef                     zOSConnectServers
   RemoteAddrGroupRef                     zOSConnectServers
   LocalPortRangeRef                     portR1
   RemotePortRangeRef                     portR3
   Userid                                       JOHNSON
   Direction                                 Outbound
   Priority                                    254
   TTLSGroupActionRef                     gAct1
   TTLSEnvironmentActionRef            eAct1
   TTLSConnectionActionRef            cAct1
}
TTLSRule                                    zCEEServerRule~3
{
   LocalAddrGroupRef                     zOSConnectServers
   RemoteAddrGroupRef                     zOSConnectServers
   LocalPortRangeRef                     portR2
   RemotePortRangeRef                     portR1
   Direction                                 Inbound
   Priority                                    253
   TTLSGroupActionRef                     gAct1
   TTLSEnvironmentActionRef            eAct2~zCEEServer
   TTLSConnectionActionRef            cAct2~zCEEServer
}
TTLSGroupAction                             gAct1
{
   TTLSEnabled                                On
   Trace                                         7
}
TTLSEnvironmentAction                    eAct1
{
   HandshakeRole                              Client
   EnvironmentUserInstance             0
   TTLSKeyringParmsRef                   keyR1
}
TTLSEnvironmentAction                    eAct2~zCEEServer
{
   HandshakeRole                              Server
   EnvironmentUserInstance             0
   TTLSKeyringParmsRef                   keyR2
}
```

```
TTLSConnectionAction                      cAct1
{
   HandshakeRole                           Client
   TTLSCipherParmsRef                     cipher1~AT-TLS__Gold
   TTLSConnectionAdvancedParmsRef   cAdv1
   Trace                                      7
}
TTLSConnectionAction                      cAct2~zCEEServer
{
   HandshakeRole                           Server
   TTLSCipherParmsRef                     cipher1~AT-TLS__Gold
   TTLSConnectionAdvancedParmsRef   cAdv2~zCEEServer
   Trace                                      7
}
TTLSConnectionAdvancedParms         cAdv1
{
   SSLv3                                      On
   SecondaryMap                             Off
}
TTLSConnectionAdvancedParms         cAdv2~zCEEServer
{
   SSLv3                                      On
   SecondaryMap                             Off
}
TTLSKeyringParms                           keyR1
{
   Keyring                                    Keyring.zCEE
}
TTLSKeyringParms                           keyR2
{
   Keyring                                    Keyring.LIBERTY
}
TTLSCipherParms                           cipher1~AT-TLS__Gold
{
   V3CipherSuites
TLS_RSA_WITH_3DES_EDE_CBC_SHA
   V3CipherSuites                           TLS_RSA_WITH_AES_128_CBC_SHA
}
IpAddrGroup                                zOSConnectServers
{
      IpAddr
      {
            Addr 192.168.141.44
      }
}
PortRange                                   portR1
{
   Port                                        1024-65535
}
PortRange                                   portR2
{
   Port                                        9120
}
PortRange                                   portR3
{
   Port                                        9173
}
```

When an API requester uses the options below while running under identity JOHNSON, AT-TLS rule *zCEEClientRule~1* will be trigged by the policy. AT-TLS will initiate a TLS handshake with the server listening on port 9120. This handshake request will trigger another AT-TLS rule, zCEEServerRule~3. This AT-TLS rule will act as the TLS server in lieu of the application server during the handshake.

```
//CEEOPTS  DD *
  POSIX(ON),
  ENVAR("BAQURI=wg31.washington.ibm.com",
  "BAQPORT=9120")
```

When an API requester is uses the options below while running under identity JOHNSON, AT-TLS rule *zCEEClientRule~2* will be trigged by the policy. AT-TLS will initiate a TLS handshake with the server listening on port 9473. The handshake will proceed using the JSSE support configured in the Liberty server where z/OS Connect is running. No inbound AT-TLS policy is triggered.

```
//CEEOPTS  DD *
  POSIX(ON),
  ENVAR("BAQURI=wg31.washington.ibm.com",
  "BAQPORT=9173")
```

Implementing a z/OS Connect EE Policies

This section provides an example of implement a z/OS Connect EE policy which determines the transaction identity under which the CICS mirror program will run.

 The first step is to create a rule set. If an HTTP header is provided in the request which matches a condition in the ruleset the value associated with the header will be checked with the rule conditions. If the header value matches one of the values in a condition, the action specified in the rule will be invoked.

In the example below (*cicsRules.xml*) if the header named *cicsMirror* is included in the request, the header value will be checked to see if it matches CSMI, MIJO, ATS0 or ATS1. If there is a match, then the CICS transaction identity will be set to the header value and the CICS mirror program DFHMIRS will be started with this value. The same applies to *cicsConnection*, if there is a match then the CICS connection reference will be set to the header value of HTTP property cicsConnection.

```
<ruleset name="CICS rules">
     <rule name="csmi-rule">
         <conditions>
             <header name="cicsMirror" value="CSMI,MIJO,ATS0,ATS1"/>
         </conditions>
         <actions>
             <set property="cicsTransId" value="${cicsMirror}"/>
         </actions>
     </rule>
     <rule name="connection-rule">
        <conditions>
            <header name="cicsConnection"   value="cscvinc,cics92,cics93"/>
        </conditions>
        <actions>
            <set property="cicsConnectionRef" value="${cicsConnection}"
        </actions>
    </rule>
 </ruleset>
```

 Next add a *zosconnect_policy* element in the *server.xml* to identify the rule set file name location and name.

```
<zosconnect_policy id="cicsPolicy"
   location="${server.config.dir}resources/zosconnect/rules">
   <ruleset file="cicsRules.xml"/>
 </zosconnect_policy>
```

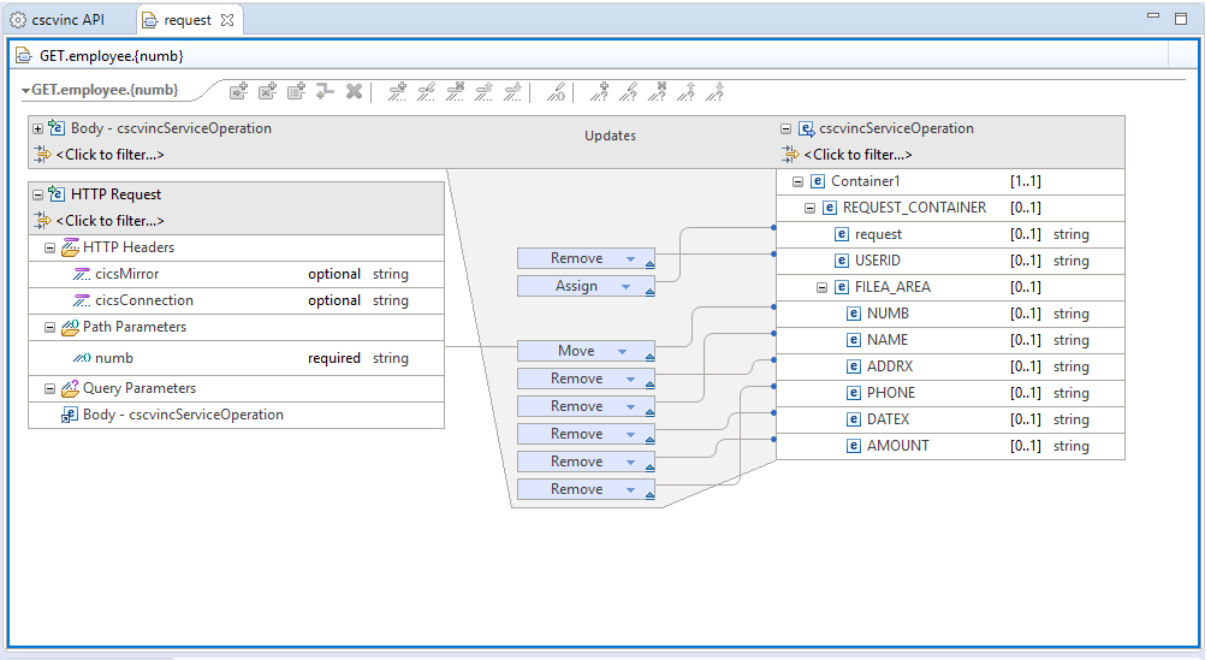
 Finally enable the policy identified in the *zosconnect_policy* element either globally in the *zosConnectApi* element or for a specific API.

```
 <!-- zosConnect APIs -->
  <zosconnect_zosConnectAPIs pollingRate="5s" updateTrigger="polled"
      policyRef="cicsPolicy"/>
```

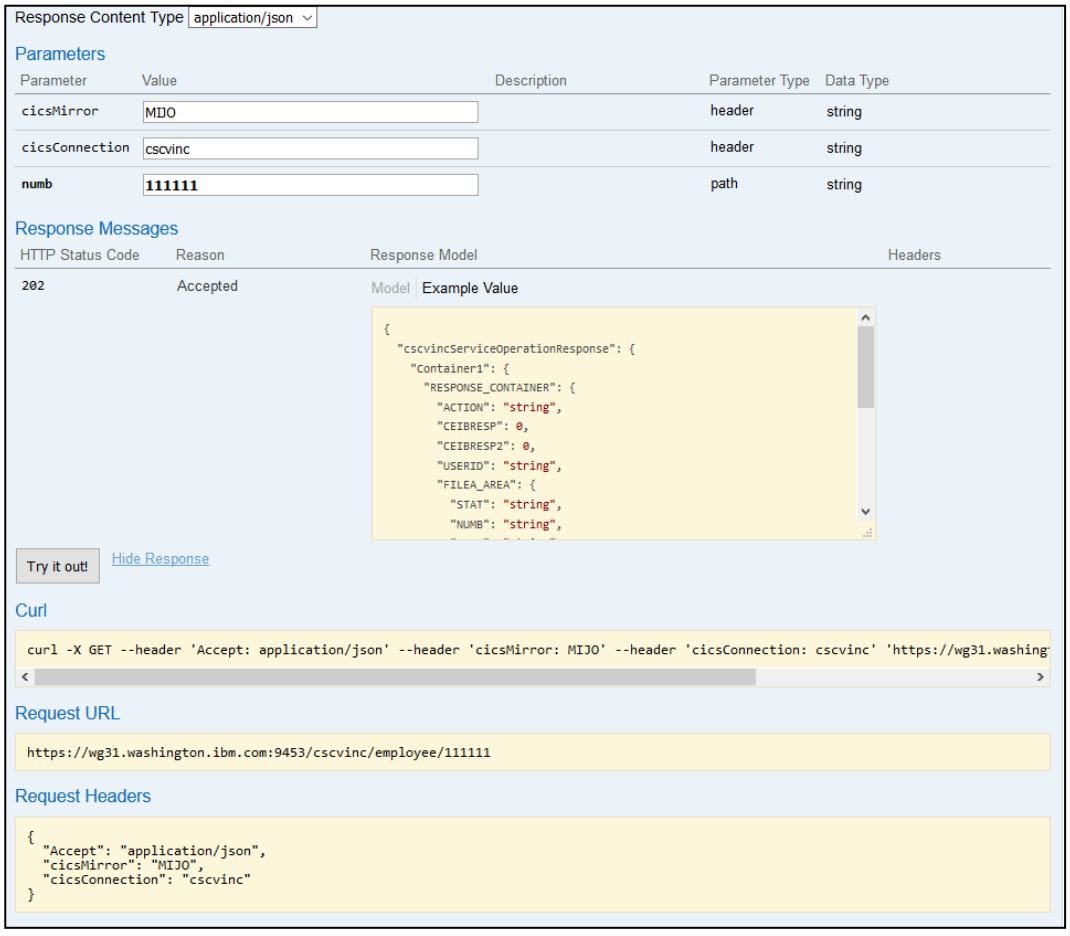
The name/value pairs added as header *cicsMirror* and *cicsConnecton* to a request as shown below

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
Basic RnJlZDpmcmVkcHdk' --header 'cicsMirror: MIJO'                  --
header 'cicsConnection: cics92'
'https://wg31.washington.ibm.com:9453/cscvinc/employee/333333'
```

Note also these header properties can be added during the mapping phases



So they will be accessible when using the Swagger-UI test interface.



Managing a z/OS Connect EE server with the Admin Center

WebSphere Liberty Profile provides an *Admin Center* feature which provide a web browser interface for viewing and/or managing a z/OS Connect EE server's configuration. Detailed information for this feature can be found at URL
https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_ui_explore.html

This section provides details on how to add this feature to the Liberty server in which z/OS Connect EE is running and how to enable security and how to enable access to the *server.xml* and any include files referenced by the *server.xml*.

Security

 If SAF security register has not been enabled, add a <*user*> configuration <user> element for each administrator identity as shown below for identity Fred.

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

 If a SAF security register is being used, define an EJBRole resource and permit read access to each administrator's identity to this EJBRole resource (see below).

```
RDEFINE EJBROLE BBGZDFLT.com.ibm.ws.management.security.resource.Administrator
OWNER(SYS1) ACC(NONE)

PERMIT BBGZDFLT.com.ibm.ws.management.security.resource.Administrator
CLASS(EJBROLE)  RESET

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

SETR RACLIST(EJBROLE) REFRESH
```

Tech Tip: The value ***BBGZDFLT*** in the above commands must match the value of attribute *profileprefix* in the existing *safCredentials* element in the server.xml.

Updates to the server.xml

The following Liberty *server.xml* updates are required.

 Add the *adminCenter-1.0* feature to the feature manager list.

```
  <featureManager>
            <feature>adminCenter-1.0</feature>
  </featureManager>
```

 To enable the updating of the *server.xml* from the web browser add these configuration elements to the server.xml.

```
<remoteFileAccess>
    <writeDir>${server.config.dir}</writeDir>
 </remoteFileAccess>
```

Tech Tip: The Admin Center can be used to view (and edit) the *server.xml*. But any files included in the *server.xml* must be accessible via the ${server.config.dir} directory structure. To address this requirement, I created a symbolic link from ${server.config.dir} to the directory containing the included files by entering OMVS command ***ln -s /wasetc/zc3lab zc3lab*** while positioned in ${server.config.dir} directory.

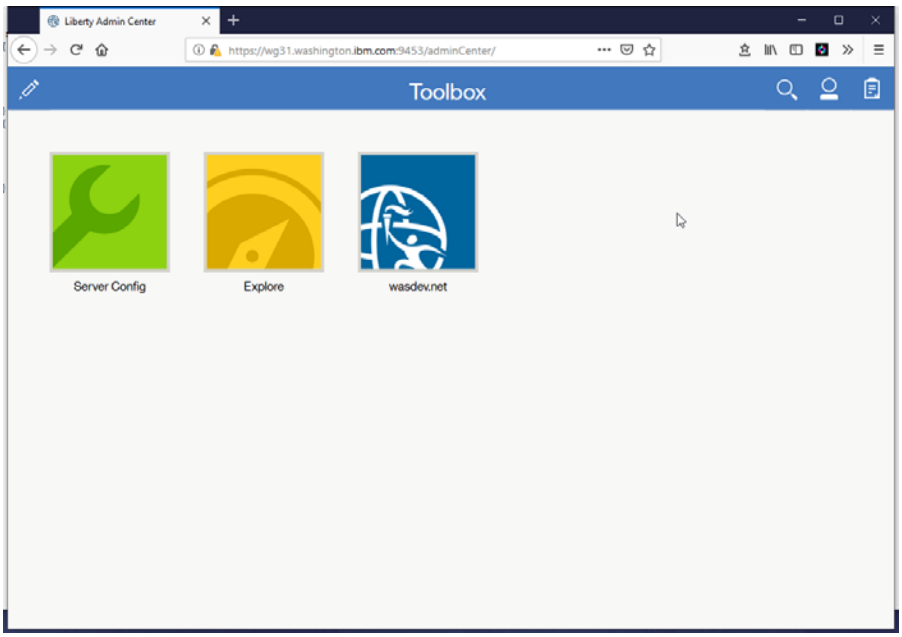
This makes files included from directory */wasetc/zc3lab* editable when included in the *server.xml* using statement *${server.config.dir}/zc3lab/* as in

<include location="${server.config.dir}/zc3lab/saf.xml" optional="true"/>

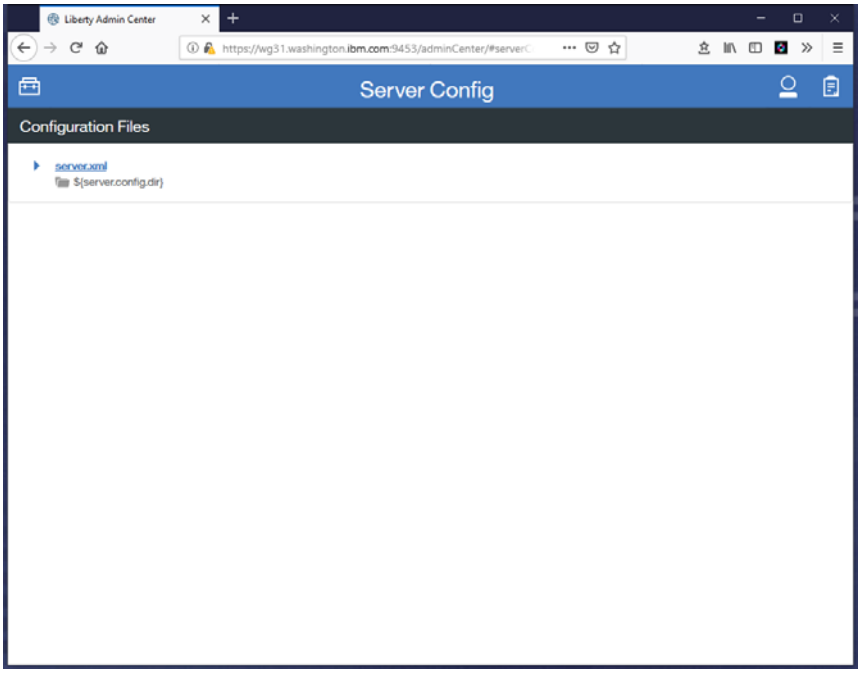
Accessing the Admin Center console

 To access the Admin Center console, enter in a web console the URI path */adminCenter*, e.g. https://wg31.washington.ibmcom:9443/adminCenter and enter a valid user identity and password. Then press the **Submit** button.

  You should see a screen like the one below. Click on the *Server Config* icon to continue.

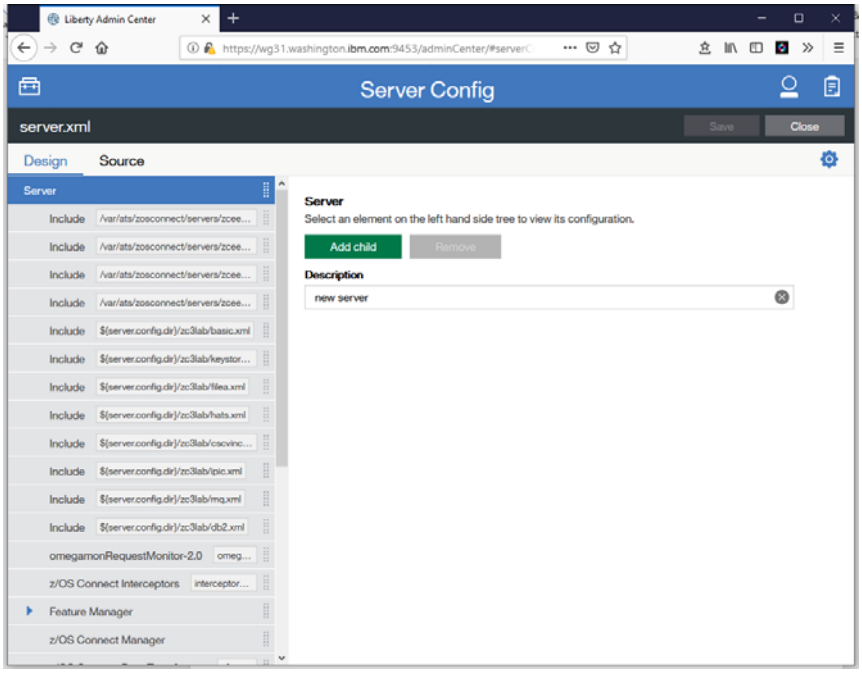


 This should display the screen below. Click on *server.xml* to continue.

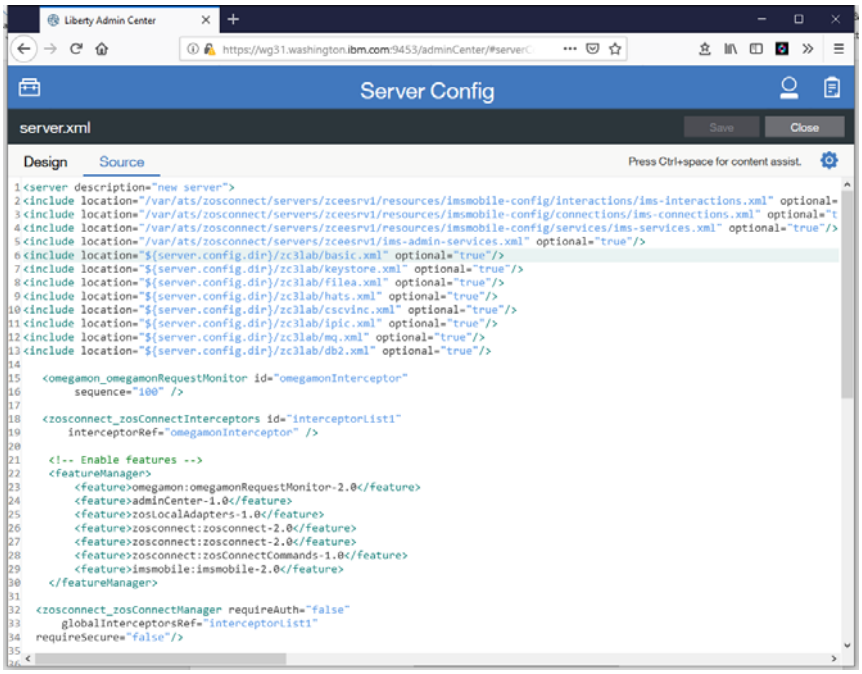


 Toggle between *Design* and *Source* to switch between views of the contents of the *server.xml.*

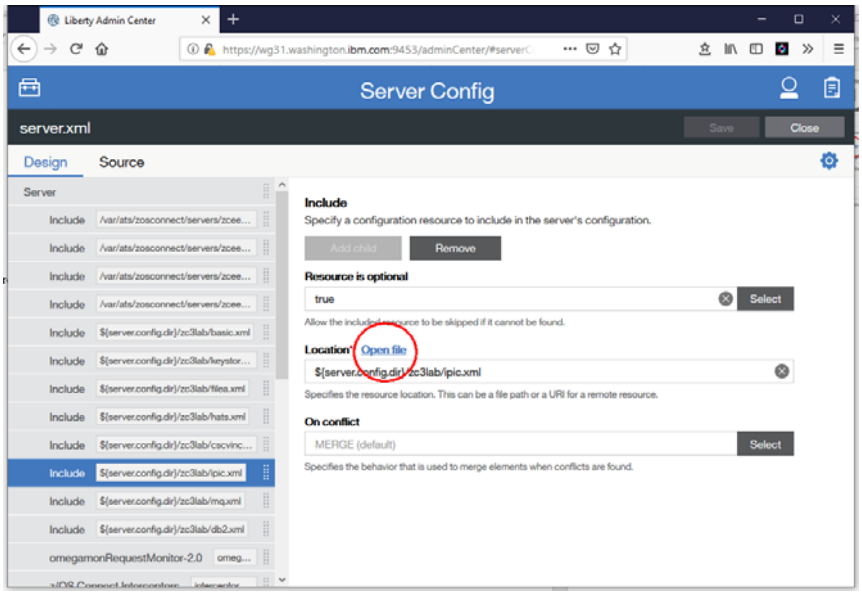
- Design View:



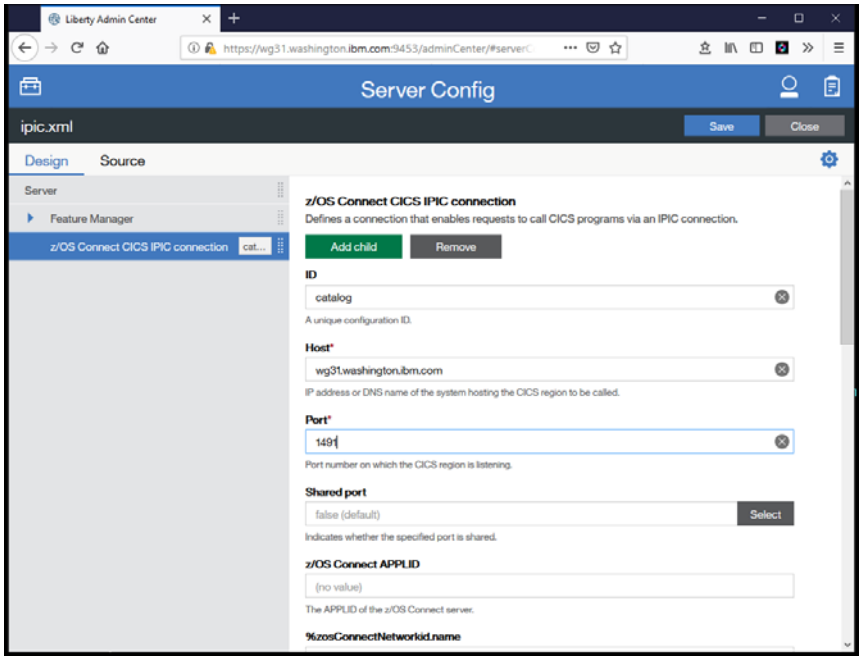
- Source View:



 Using the *Design* view, you can select an include file and use the Open file option (see below) to display the contents of the file.



 On this screen an administrator can make changes to the configuration.



This provided a basic introduction to using the Admin Center console.

Alternatives to using CEEOPTS DD input for API Requesters

LE runtime options are used to pass parameters to the z/OS Connect EE (zCEE) API requester communication stub when an MVS batch or IMS API requester application invokes an external API using the zCEE API requester feature.

These LE runtime options enable a POSIX compatible runtime LE enclave (required for the stub) and environment variables which provide the host name on which a zCEE server resides and the port on which the server is listening for inbound request. Also present are environment variables that provide security credentials used for authenticating to the zCEE server.

These security credentials (and perhaps the host and port information also) are sensitive and probably it is not desirable to have these credentials exposed in clear text in the JCL of the job used to execute an API requester application, see the CEEOPTS DD statement input as shown below.

```
//GET  EXEC PGM=GETAPI,PARM='111111'
//STEPLIB  DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
//         DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEOPTS  DD *
 POSIX(ON),
 ENVAR("BAQURI=wg31.washington.ibm.com",
 "BAQPORT=9120",
"BAQUSERNAME=USER1",
"BAQPASSWORD=USER1")
//
```

A possible way to avoid specifying these LE runtime options in the JCL is to take advantage of an LE customization option where LE runtime options can be stored in a load module and obtained at execution time either by dynamically loading this module or having the module statically linked into the application load module.

Dynamically loading LE runtime options overrides is done be creating a load module named CEEROPT. This load module is then place in either the JOBLIB or STEPLIB concatenation sequences. This technique provides a solution where multiple sets of API requester applications can access the same load library containing the CEEROPT module and share the same set of LE override options concurrently. Statically linking a LE runtime options override module is done by creating a load module names CEEUOPT and then directly linking CEEUOPT into the API requester application load module during its linkage editing process.

There are advantages to both methods. For example, when using the dynamic loaded CEEROPT module a change to a user name or password simply means recreating the load module once and all application have immediate access to the change information the next time they are executed Changing the statically linked module CEEUOPT means a change to its runtime options requires relinking all

applications once the CEEUOPT module is updated. Statically linking the runtime options into the application load modules does provide runtime options isolation.

These load modules are described in the LE Customization Guide.

Creating a CEEROPT module

When dynamic loading is enabled (e.g. *SETCEE CEEROPT,ALL*), the LE runtime will check to see if a CEEROPT load module is accessible in either the JOBLIB or STEPLIB concatenation sequences. If a module with this name found, then this module will be used to provide overrides for system wide default LE runtime options. This allows the same CEEROPT module to be shared across multiple instances of API requester client application. The CEEROPT load module is created by assembling a CEEXOPT macro and linking it into a load library.

```
//ASSEM EXEC PGM=ASMA90,PARM='DECK,NOOBJECT'
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&TEMPOBJ(CEEROPT),DISP=(,PASS),UNIT=SYSDA,
// SPACE=(TRK,(1,1,1)),DCB=(BLKSIZE=3120,LRECL=80,DSORG=PO)
//SYSLIB DD DSN=CEE.SCEEMAC,DISP=SHR
//       DD DSN=SYS1.MACLIB,DISP=SHR
//SYSIN DD *
CEEROPT  CSECT
CEEROPT  AMODE ANY
CEEROPT  RMODE ANY
         CEEXOPT POSIX=((ON),OVR),                                    +
               ENVAR=(('BAQURI=wg31.washington.ibm.com',              +
               'BAQPORT=9120',                                        +
               'BAQUSERNAME=USER1',                                   +
               'BAQPASSWORD=USER1'),OVR),                             +
               RPTOPTS=((ON),OVR)
//LKED  EXEC  PGM=IEWL,
//         PARM='NCAL,RENT,LIST,XREF,LET,MAP,SIZE=(9999K,96K)'
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(5,5))
//SYSLMOD  DD  DSNAME=USER1.ZCEE.LOADLIB,DISP=SHR
//SYSLIB   DD  DSN=&&TEMPOBJ,DISP=(OLD,PASS)
//SYSLIN   DD  *
 INCLUDE SYSLIB(CEEROPT)
  ENTRY  CEEROPT
   ORDER CEEROPT
 NAME CEEROPT(R)
/*
```

Tech-Tip: In the above example the plus signs (+) are in column 72 of the macro's source. The CEEXOPT macro starts in column 10 and the continuation lines start in column 16. The RUNOPTS options displays the LE runtime options as they are set at execution time in the job's output.

This load library (USER1.ZCEE.LOADLIB) can then be placed in the JOBLIB or STEPLIB concatenation list of the JCL used to execute the API requester client applications (see below).

Compiling and linking an API requester application

The JCL used to compile and link edit a API requester client application does not change

```
//COMPILE EXEC IGYWCL,LNGPRFX=IGY620,PARM.COBOL='NODYNAM'
//COBOL.SYSIN  DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(GETAPI)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
//             DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(GETAPI)
//LKED.SYSLIB  DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
//LKED.BAQLIB  DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//LKED.SYSIN   DD *
  INCLUDE BAQLIB(BAQCSTUB)
//
```

Only the BAQCSTUB needs to be include in the linkage process.

Creating a CEEUOPT module

Statically linking the runtime options module for each individual API requester application means that the CEEUOPT module is linked directly into the API requester load module. The CEEUOPT load module is created by assembling a CEEXOPT macro and then linking it into a load library

```
//ASSEM EXEC PGM=ASMA90,PARM='DECK,NOOBJECT'
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&TEMPOBJ(CEEUOPT),DISP=(,PASS),UNIT=SYSDA,
// SPACE=(TRK,(1,1,1)),DCB=(BLKSIZE=3120,LRECL=80,DSORG=PO)
//SYSLIB DD DSN=CEE.SCEEMAC,DISP=SHR
//       DD DSN=SYS1.MACLIB,DISP=SHR
//SYSIN DD *
CEEUOPT  CSECT
CEEUOPT  AMODE ANY
CEEUOPT  RMODE ANY
         CEEXOPT POSIX=(ON),                                          +
               ENVAR=('BAQURI=wg31.washington.ibm.com',               +
               'BAQPORT=9120',                                        +
               'BAQUSERNAME=Fred',                                    +
               'BAQPASSWORD=fredpwd')
//LKED    EXEC  PGM=IEWL,
//         PARM='NCAL,RENT,LIST,XREF,LET,MAP,SIZE=(9999K,96K)'
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(5,5))
//SYSLMOD  DD  DSN=USER1.ZCEE.LOADLIB,DISP=SHR
//SYSLIB   DD  DSN=&&TEMPOBJ,DISP=(OLD,PASS)
//SYSLIN   DD  *
 INCLUDE SYSLIB(CEEUOPT)
  ENTRY  CEEUOPT
   ORDER CEEUOPT
 NAME CEEUOPT(R)
/*
```

Tech-Tip: In the above example the plus signs (+) are in column 72 of the macro's source. The CEEXOPT macro starts in column 10 and the continuation lines start in column 16.

Compiling and linking an API requester application with static override

The JCL to compile and link API request module in this case does change to add ORDER and INCLUDE statements for the CEEUOPT module.

```
//COMPILE EXEC IGYWCL,LNGPRFX=IGY620,PARM.COBOL='NODYNAM'
//COBOL.SYSIN  DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(GETAPI)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
//             DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(GETAPI)
//LKED.SYSLIB  DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
//LKED.BAQLIB  DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//LKED.SYSIN   DD *
 ORDER CEESTART
 INCLUDE SYSLIB(CEEUOPT)
 INCLUDE BAQLIB(BAQCSTUB)
//
```

Updated JCL for executing the API request application

When using either dynamic or static LE option overrides, the JCL to execute the API requester application is changed to remove the CEEOPTS DD statement. Otherwise the JCL is the same.

```
//COMPILE EXEC PGM=GETAPI,PARM='111111'
//STEPLIB  DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
//         DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//
```

Troubleshooting RACF issues with Liberty and z/OS Connect servers

This section documents some of the more common RACF related resource and/or configuration issues. This is not an all-encompassing list of issues or their causes but perhaps the information contained here will help a reader identity and address their specific issue or situation.

Liberty Server Startup Errors

This first set of messages appear in the *messages.log* file at server startup and indicate insufficient access to the angel and/or other required RACF SERVER resources.

The RACF command that permits the required access is provided for each message. In these examples ATSGRP is a RACF group for the RACF identities under which the z/OS Connect Liberty servers ae running. Group ATSUSERS is a RACF group of identities authorized to use an instance of z/OS Connect.

- ***CWWKB0117W: The ZCEE angel process is not available. No authorized services will be loaded. The reason code is 5.***

Cause/Solution: The server is trying to access a named angel but no angel with the specified name (e.g., *zCEE*) is active. Start an angel with this name (*S BBGZANGL,**NAME=ZCEE***) or change the *com.ibm.ws.zos.core.angelName* Java option to provide the name of an active angel.

- ***CWWKB0117W: The angel process is not available. No authorized services will be loaded. The reason code is 4.***

Cause/Solution: The server is trying to access the default angel, but the default angel is not active. Start a default angel (e.g. one with no name).

```
CWWKB0079I THE ANGEL BUILD LEVEL IS 19.0.0.9 20190905-0519 / 2019.9.0.0 20190905-0519
CWWKB0069I INITIALIZATION IS COMPLETE FOR THE ZCEE ANGEL PROCESS.
```

- ***CWWKB0118W: This server is not authorized to connect to the ZCEE angel process. No authorized services will be loaded.***

Cause/Solution: The RACF identity under which the server is executing does not have sufficient (READ) access to the RACF SERVER resource protecting the angel, be sure the appropriate profile is define and permit the Liberty server's RACF identity (group or user) to have READ access to this profile.

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

▪ ***CWWKB0117W: The ZCEE angel process is not available. No authorized services will be loaded. The reason code is 4,104.***
CWWKB0115I: This server is not authorized to load module bbgzsafm. No authorized services will be loaded.

Cause/Solution: The server registration with the angel failed because the server was not authorized to the BBGZSAFM resource. Permit READ access to SERVER resource BBG.AUTHMOD.BBGZSAFM to the RACF identity under which the server is executing.

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

The next set of messages are related to required features. Access to these features require READ access to various SERVER resources.

▪ ***CWWKB0104I: Authorized service group LOCALCOM is not available.***

```
PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS(SERVER) ACCESS(READ) ID(ATSGRP)
```

▪ ***CWWKB0104I: Authorized service group PRODMGR is not available.***

```
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(ATSGRP)
```

▪ ***CWWKB0104I: Authorized service group SAFCRED is not available.***

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

▪ ***CWWKB0104I: Authorized service group TXRRS is not available.***

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

• ***CWWKB0104I: Authorized service group ZOSAIO is not available.***

```
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSAIO CLASS(SERVER) ACCESS(READ) ID(ATSGRP)
```

▪ ***CWWKB0104I: Authorized service group ZOSDUMP is not available.***

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

- ***CWWKB0104I: Authorized service group ZOSWLM is not available***.

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

- ***CWWKB0104I: Authorized service group WOLA is not available.***

```
PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS(SERVER)ACCESS(READ) ID(ATSGRP)
```

- ***CWWKB0104I: Authorized service group CLIENT.WOLA is not available***

```
PERMIT BBG.AUTHMOD.BBGZSCFM CLASS(SERVER) ACCESS(READ) ID(ATSGRP)
PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS(SERVER) ACCESS(READ) ID(ATSGRP)
```

.

Messages related to enabling RACF security

RACF enablement messages will sometimes appear in the SYSLOG and/or console messages, but the real issue is usually identified in the *messages.log* file.

- ***BPXP015I HFS PROGRAM programName IS NOT MARKED PROGRAM CONTROLLED***

The BPXP015I message will appear in the SYSLOG output along with other BPX messages. The *programName* value in the message will vary.

```
BPXP015I HFS PROGRAM /usr/lib/java_runtime/libifaedjreg64.so IS NOT  MARKED
PROGRAM CONTROLLED.
BPXP014I ENVIRONMENT MUST BE CONTROLLED FOR DAEMON (BPX.DAEMON)
PROCESSING.
BPXP015I HFS PROGRAM /usr/lib/java_runtime/libifaedjreg64.so IS NOT MARKED
PROGRAM CONTROLLED.
BPXP014I ENVIRONMENT MUST BE CONTROLLED FOR DAEMON (BPX.DAEMON)
PROCESSING
```

Generally, the BPX messages in the SYSLOG are misleading. The useful information will appear concurrently in the *messages.log* file as shown below:

CWWKS2930W: A SAF authentication attempt using authorized SAF services was rejected because the server is not authorized to access the APPL-ID ATSZDFLT. Authentication will proceed using unauthorized SAF services.

CWWKS2933E: The username and password could not be checked because the BPX.DAEMON profile is active, and the address space is not under program control.

CWWKS1100A: Authentication did not succeed for user ID Fred. An invalid user ID or password was specified.

CWWKS2933E: The username and password could not be checked because the address space is not under program control.

The issue is caused a missing RACF resource or lack of access to a RACF APPL resource identified in message *CWWKS2930W*. In this example the server.xml file contained the *safCredential* configuration element shown below:

```
<safCredentials unauthenticatedUser="ZCGUEST" Prefix="ATSZDFLT" />
```

The value used for *profilePrefix* (the default value is *BBGZDFLT*) must be defined as a RACF APPL resource. Note that all RACF identities that will be access this server must have READ access to this APPL resource. A server resource BBG.SECPFX.***ATSZDFLT*** must also all be defined with the Liberty server's RACF identity having READ access to this resource.

```
RDEFINE APPL ATSZDFLT UACC(NONE) OWNER(SYS1)
PERMIT ATSZDFLT CLASS(APPL) ACCESS(READ) ID(ZCGUEST,ATSGRP)
RDEFINE SERVER BBG.SECPFX.ATSZDFLT UACC(NONE)
PERMIT  BBG.SECPFX.ATSZDFLT CLASS(SERVER) ACCESS(READ)
ID(ATSGRP)
```

The following messages will primarily appear in the messages.log file.

- ***CWWKS2909E: A SAF authentication or authorization attempt was rejected because the server is not authorized to access the following SAF resource: APPL-ID ATSZDFLT. Internal error code 0x03008108.***

  This identity under which the server is running does not have READ access to the APPL resource ATSZDFLT. Connect the user to a group which has READ access or provide explicit access using the command shown below:

```
PERMIT ATSZDFLT CLASS(APPL) ACCESS(READ) ID(ATSSERV) ACC(READ)
```

- ***CWWKS2911E: SAF Service RACROUTE_AUTH did not succeed because the resource profile ATSZDFLT.zos.connect.access.roles.zosConnectAccess in class EJBROLE does not exist. SAF return code 0x00000004. RACF return code 0x00000004. RACF reason code 0x00000000.***

The RACF return and reason code indicate that this RACF EJBROLE resource has not been defined. The value of the *profilePrefix* specified in the *safCredential* configuration element is prepended to the role *zos.connect.access.roles.zosConnectAccess* to form the name of the RACF EJBROLE resource which controls access to this server. The EJBRole is defined to RACF as shown below. Note that all authorized z/OS Connect users will need READ access to this EJBROLE.

```
RDEFINE EJBROLE ATSZDFLT.zos.connect.access.roles.zosConnectAccess
OWNER(SYS1) UACC(NONE)
Then permit access to this resource to all authorized users
PERMIT ATSZDFLT.zos.connect.access.roles.zosConnectAccess CLASS(EJBROLE)
ID(ATSUSERS,ATSSERV) ACCESS(READ)
```

- ***CWWKS2907E: SAF Service IRRSIA00_CREATE did not succeed because user user1 has insufficient authority to access APPL-ID ATSZDFLT. SAF return code 0x00000008. RACF return code 0x00000008. RACF reason code 0x00000020.***

CWWKS1100A: Authentication did not succeed for user ID user1. An invalid user ID or password was specified.

This user does not have READ access to the APPL resource ATSZDFLT. Connect the user to the *ATSUSERS* group or provide explicit access using the command shown below:

```
PERMIT ATSZDFLT CLASS(APPL) ACCESS(READ) ID(USER1) ACC(READ)
```

- *MVS console message:*

ICH408I USER(USER1) GROUP(SYS1   ) NAME(WORKSHOP USER1

ATSZDFLT.zos.connect.access.roles.zosConnectAccess

CL(EJBROLE)

INSUFFICIENT ACCESS AUTHORITY

ACCESS INTENT(READ) ACCESS ALLOWED(NONE)

This user does not have READ access to the EJBROLE resource. Connect the user to the *ATSUSERS* group or provide explicit access using the command shown below:

```
PERMIT ATSZDFLT.zos.connect.access.roles.zosConnectAccess  CLASS(EJBROLE)
ID(USER1) ACCESS(READ)
```

▪ ***FFDC1015I: An FFDC Incident has been created: "java.io.IOException : R_datalib (IRRSDL00) error: profile for ring not found (8, 8, 84) com.ibm.ws.ssl.config.WSKeyStore$1 do_getKeyStore" at ffdc_19.11.20_13.28.35.0.log***

Cause/Solution: The key ring identified in the keystore configuration element has not been defined in RACF. Define and configure the key ring, e.g. *Liberty.KeyRing*.

```
<ssl id="DefaultSSLSettings"
  clientAuthentication="false"
  clientAuthenticationSupported="true"
  keyStoreRef="CellDefaultKeyStore"
  trustStoreRef="CellDefaultKeyStore" />
<keyStore id="CellDefaultKeyStore"
  location="safkeyring:///Liberty.KeyRing"
  password="password" type="JCERACFKS"
  fileBased="false" readOnly="false" />
```

▪***CWWKO0801E: Unable to initialize SSL connection. Unauthorized access was denied or security settings have expired. Exception is javax.net.ssl.SSLHandshakeException: no cipher suites in common***

Cause/Solution: There may be many causes for this issue but first confirm the RACF identity under which the server is running has READ access to FACILITY resources *IRR.DIGTCERT.LISTRING* and *IRR.DIGTCERT.LIST*. The first resource gives the identity access to their own key ring and the second allows access to the certificates.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(ATSSERV) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(ATSGRP)  ACCESS(READ)
```

Messages related to exchanging digital certificates (TLS)

This set of messages may appear when connecting to a server in a browser or when invoking an outbound API request. With a few exceptions most of TLS errors will require a review of a trace.

Enable the *traceSpecification* shown below and review the generated trace for these

```
<logging traceSpecification="com.ibm.ws.security.*=all:
       SSLChannel=all:SSL=all:zosConnectSaf=all"/>
```

This will generate a *trace.out* file in the *logs* subdirectory. This trace will provide details about the key ring and certificates involved in the handshake. There is a wealth of information about the flow between the client and server endpoints Review this trace for exceptions. The following exceptions are the ones most commonly experienced.

- ***Error occurred during a read, exception:javax.net.ssl.SSLHandshakeException: null cert chain***

This exception occurs when the server configuration set to require client certificates (*clientAuthenication="true")* and the client had no certificate to provide and no alternative authentication method was available.

This can occur when a browser tries to connection to the administrative interface and the local key store does not have a valid personal certificate. The browser will display a message that a ***Secure Connection Failed*** and a message like the one below:

An error occurred during a connection to wg31.washington.ibm.com:9443. PR_END_OF_FILE_ERROR

- ***Error occurred during a read, exception:javax.net.ssl.SSLException: Received fatal alert: bad_certificate error (handshake), vc=1083934466***

Caught exception during unwrap, javax.net.ssl.SSLException: Received fatal alert: bad_certificate

This is usually caused when the client certificate presented to the server did not have a valid CA certificate for the client's personal certificate in the server's trust store key ring.

- ***FFDC1015I: An FFDC Incident has been created: "java.io.IOException: Failed validating certificate paths com.ibm.ws.ssl.config.WSKeyStore$1 do_getKeyStore" at ffdc_19.12.04_20.51.47.0.log***

This is can occur when the CA certificate used to sign the server's personal certificate was not connected to the server's local trust store (key ring on z/OS).

▪***java.io.IOException: IOException invoking https://132.25.33.351:9443/employees/John?validated=true: HTTPS hostname wrong: should be <132.25.33.351>***

Cause/resolution: In this situation the endpoint for the outbound API request was configured to use an IP address rather than a hostname. This should not be an issue unless an exchange of digital certificates is required.

The trace showed that during the handshake process the outbound API provider server's certificate had a common name (CN) which specified the hostname of the TCPIP stack where the API resided. This hostname was not known (e.g. DNS-resolvable) on the TCPIP stack where the z/OS Connect server was executing. This meant that communications back to the API requester's TCPIP stack based on the hostname was not possible which caused the IO exception. The best solution would be to use the host name in the server.xml configuration rather than the IP address and either add an entry to the local TCPIP stack's hostname (e.g. hosts) file for the IP address and hostname or add an entry to the DNS servers used by this TCPIP stack.

WebSphere Local Optimized Adapter Error Messages

In this section the following configuration for the *zosLocalAdapters* configuration element are in the server.xml file of the Liberty server.

```
<zosLocalAdapters wolaGroup="ZCEESRVR"
    wolaName2="ZCEESRVR"
    wolaName3="ZCEESRVR"/>

<connectionFactory id="wolaCF"
    jndiName="eis/ola">
    <properties.ola/>
</connectionFactory>
```

- *Call to BBOA1REG failed with Return Code = 00000012 Reason Code = 00000016*

There are several causes for this message but the most common is that the RACF CBIND that is used to managed connection to the WOLA interface is not defined. Problem isolation begins by reviewing the Liberty server's messages.log file to determine if the message below appears:

- *CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with the Liberty profile server using the following name: ZCEESRVR ZCEESRVR ZCEESRVR*

If this message does not appear and there are no other WOLA related error messages, try confirming the CBIND resource for this name has been defined and the identity associated with the client request has READ access. If not defined the resource as shown below:

```
RDEFINE CBIND BBG.WOLA.ZCEESRVR.ZCEESRVR.ZCEESRVR UACC(NONE)
OWNER(SYS1)
PERMIT BBG.WOLA.ZCEESRVR.ZCEESRVR.ZCEESRVR  CLASS(CBIND)
ACCESS(READ) ID(USER1,CICSX)
```

- ***Call to BBOA1SRV failed - Return Code = 00000012 Reason Code = 00000014***

This message indicates that the RACF identity of the client does not have access to the CBIND resource protecting the WOLA interface. Ensure that the user ID is authorized to the CBIND SAF class for the requested WOLA server as shown above.

- ***Call to BBOA1REG failed - Return Code = 00000012 Reason Code = 00000088***

The most common cause is that no Liberty server has successfully registered a WOLA channel with the names specified by the client. Problem isolation begins by reviewing the Liberty server's messages.log file to determine if the message below appears:

- ***CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with the Liberty profile server using the following name: ZCEESRVR ZCEESRVR ZCEESRVR***

If this message has not appeared, then review the messages.log file and resolve any WOLA related issues identified in the Liberty server's configuration and/or RACF profile access and then try to restart the client.

End of WP102724