(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

## Document Overview ................................................................. 6
Program numbers and FMIDs ....................................................... 8
Service and Maintenance URLs ................................................ 8
The z/OS Connect EE Knowledge Center URL .............................. 8
The WebSphere Application Server for z/OS Liberty Knowledge Center URL .............................................. 8
IBM developerWorks articles URL ............................................. 8
Additional IBM Support web pages .......................................... 8

## Installation and Initial Setup ...................................................... 9
SMP/E install of z/OS Connect EE .............................................. 10
Essential prerequisites .............................................................. 10
OMVS Ownership/Permissions Bits Considerations .................. 10
  Use of Surrogate Access..................................................... 11
  Permit access to Unix Privileges ....................................... 11
Confirm that a Java runtime can be created ............................. 12
Liberty angel Considerations .................................................. 14
  Named angels .................................................................... 14
Post SMP/E configuration steps .............................................. 15
SAF Resources ........................................................................ 18
  SAF Groups and Server IDs ............................................. 18
  SAF STARTED profiles .................................................. 19
  SAF SERVER and FACILITY profiles ................................ 19
z/OS Connect Server creation ................................................. 21
  TCP ports and host element ............................................ 24
Start a z/OS Connect EE server .............................................. 25
Setup of basic security .......................................................... 27
Installing the z/OS Connect EE V3.0 tooling ......................... 34
  Installing an Eclipse runtime platform ............................... 34
  Installing the z/OS Connect EE V3.0 API Toolkit in an Eclipse environment ........................................ 34
  Installing the z/OS Connect EE V3.0 API Toolkit in an IBM Installation Manager environment .......... 36
Checkpoint: status at this point .............................................. 38
Open IBM z/OS Explorer for z/OS and connect to the z/OS Connect EE server .................................... 39

## CICS RESTful APIs ................................................................. 43
Adding IPIC support to a z/OS Connect server ......................... 43
Setup of IPIC support in a CICS region .................................... 44
Developing RESTful Services for CICS .................................... 45
Test the Services ..................................................................... 45
  Security and CICS .......................................................... 48

## IMS TM RESTful APIs ............................................................. 49
Adding IMS Connect support to a z/OS Connect server .......... 49
Install the IMS Phone Sample in the IMS control region ......... 50
Verify the IMS Service Provider ............................................ 50
  Using Postman .............................................................. 52
  Using cURL ................................................................. 55
IMS definitions (connections and interactions) ....................... 56
Developing RESTful Services for an IMS transaction .......... 60
Test the Services .................................................................... 60
  Security and IMS TM ..................................................... 62

## IMS DB RESTful APIs ............................................................. 63
Adding IMS Database support to a z/OS Connect server ......... 63
  A Review of the IMS artifacts .......................................... 64

© 2016, 2020 IBM Corporation - 3 - Version Date: Tuesday, August 18, 2020
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing RESTful Services for an IMS database</td>
<td>65</td>
</tr>
<tr>
<td>Test the Services</td>
<td>65</td>
</tr>
<tr>
<td>Security and IMS DB</td>
<td>66</td>
</tr>
<tr>
<td><strong>Db2 RESTful APIs</strong></td>
<td>67</td>
</tr>
<tr>
<td>Creating Db2 REST Services</td>
<td>68</td>
</tr>
<tr>
<td>Adding Db2 REST support to a z/OS Connect server</td>
<td>71</td>
</tr>
<tr>
<td>Developing RESTful Services for Db2 Native REST Services</td>
<td>72</td>
</tr>
<tr>
<td><strong>Db2 Stored Procedure Considerations</strong></td>
<td>72</td>
</tr>
<tr>
<td>Test the Services</td>
<td>74</td>
</tr>
<tr>
<td>Security and Db2</td>
<td>78</td>
</tr>
<tr>
<td><strong>IBM MQ RESTful APIs</strong></td>
<td>79</td>
</tr>
<tr>
<td>Adding the IBM MQ Service provider support to a z/OS Connect server</td>
<td>79</td>
</tr>
<tr>
<td>Adding JMS resources to the z/OS Connect EE configuration</td>
<td>79</td>
</tr>
<tr>
<td>Developing RESTful Services for MQ</td>
<td>80</td>
</tr>
<tr>
<td>Test the Services</td>
<td>80</td>
</tr>
<tr>
<td>Security and MQ</td>
<td>81</td>
</tr>
<tr>
<td><strong>Security Topics</strong></td>
<td>82</td>
</tr>
<tr>
<td>Beyond the simple server.xml security elements</td>
<td>82</td>
</tr>
<tr>
<td>Turning off SSL and/or Authentication</td>
<td>82</td>
</tr>
<tr>
<td>Using SAF for controlling z/OS Connect EE access</td>
<td>87</td>
</tr>
<tr>
<td>Using RACF for TLS and trust/key store management</td>
<td>91</td>
</tr>
<tr>
<td>Using client certificates for authentication</td>
<td>97</td>
</tr>
<tr>
<td>RACF Certificate Mapping and Filtering</td>
<td>103</td>
</tr>
<tr>
<td>CICS Identity Propagation</td>
<td>104</td>
</tr>
<tr>
<td>IMS TM PassTickets</td>
<td>106</td>
</tr>
<tr>
<td>IMS DB PassTickets</td>
<td>107</td>
</tr>
<tr>
<td>Db2 PassTickets</td>
<td>108</td>
</tr>
<tr>
<td>Db2 REST services security</td>
<td>109</td>
</tr>
<tr>
<td>MQ services security</td>
<td>110</td>
</tr>
<tr>
<td>MQ TLS security</td>
<td>110</td>
</tr>
<tr>
<td>z/OS Connect and AT-TLS</td>
<td>112</td>
</tr>
<tr>
<td>Troubleshooting RACF issues with Liberty and z/OS Connect servers</td>
<td>117</td>
</tr>
<tr>
<td><strong>Liberty Server Startup Errors</strong></td>
<td>117</td>
</tr>
<tr>
<td>Messages related to enabling RACF security</td>
<td>119</td>
</tr>
<tr>
<td>Messages related to exchanging digital certificates (TLS)</td>
<td>124</td>
</tr>
<tr>
<td>WebSphere Optimized Local Adapter</td>
<td>126</td>
</tr>
<tr>
<td>WOLA Security</td>
<td>126</td>
</tr>
<tr>
<td>WOLA Error Messages</td>
<td>126</td>
</tr>
<tr>
<td><strong>Miscellaneous Topics</strong></td>
<td>128</td>
</tr>
<tr>
<td>Testing z/OS Connect Services Using Postman</td>
<td>128</td>
</tr>
<tr>
<td>Testing z/OS Connect Services Using cURL</td>
<td>134</td>
</tr>
<tr>
<td>Implementing a z/OS Connect EE Policies</td>
<td>136</td>
</tr>
<tr>
<td>Managing CORS updates</td>
<td>139</td>
</tr>
<tr>
<td>Liberty Environment Variables</td>
<td>139</td>
</tr>
<tr>
<td>Managing a z/OS Connect EE server with the Admin Center</td>
<td>141</td>
</tr>
<tr>
<td>Alternatives to using CEEOPTS DD input for API Requesters</td>
<td>146</td>
</tr>
<tr>
<td>Controlling dynamic updates</td>
<td>150</td>
</tr>
<tr>
<td>z/OS Connect and Data Virtualization Manager</td>
<td>151</td>
</tr>
<tr>
<td>Sample JCL</td>
<td>155</td>
</tr>
<tr>
<td>Base64 Encoding and Swagger UI</td>
<td>159</td>
</tr>
</tbody>
</table>
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 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:

<table>
<thead>
<tr>
<th>Topic and Objective</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation and Initial Setup</td>
<td>9</td>
</tr>
<tr>
<td>Before you can begin composing services and APIs, you must install z/OS Connect EE, set up the server runtime, and perform a few other tasks. This section will guide you through that process and provide simple validation tests to insure you are on the right track.</td>
<td></td>
</tr>
<tr>
<td>CICS RESTful APIs</td>
<td>43</td>
</tr>
<tr>
<td>If your initial focus is CICS as the backend, then this section will guide you through the setup. Then a step-by-step example of enabling SARs and APIs to the CICS catalog manager sample is provided via an external link.</td>
<td></td>
</tr>
<tr>
<td>IMS TM RESTful APIs</td>
<td>49</td>
</tr>
<tr>
<td>If your initial focus is IMS transactions as the backend, then this section will guide you through the setup and validation of the IMS service provider. Then a step-by-step example of enabling SARs and APIs the Phone Book sample is provided via an external link.</td>
<td></td>
</tr>
<tr>
<td>IMS DB RESTful APIs</td>
<td>63</td>
</tr>
<tr>
<td>If your initial focus is IMS database as the backend, then this section will guide you through the setup and validation of the IMS database service provider. Then a step-by-step example of enabling SARs and APIs the Phone Book data base.</td>
<td></td>
</tr>
<tr>
<td>Db2 RESTful APIs</td>
<td>67</td>
</tr>
<tr>
<td>If your initial focus is Db2 as the backend, then this section will guide you through the setup and validation of the Db2 REST services. Then a step-by-step example of developing APIs to access some common Db2 requests via an external link.</td>
<td></td>
</tr>
<tr>
<td>MQ RESTful APIs</td>
<td>79</td>
</tr>
<tr>
<td>If your initial focus is MQ as the backend, then this section will guide you through the setup and validation of the MQ service provider. Then a step-by-step example of configuring the MQ Service provider in z/OS Connect and developing APIs to access two-way and one-way MQ services via an external link.</td>
<td></td>
</tr>
<tr>
<td>Security Topics</td>
<td>82</td>
</tr>
<tr>
<td>This section is where we collect information security topics. Topics that cover enabling security using certificates, enabling security between z/OS subsystems, etc.</td>
<td></td>
</tr>
</tbody>
</table>
Miscellaneous Topics
This section is where we collect information on various topics that is of interest but is not appropriate to be included in line with the step-by-step instructions. We point to topics in this section from elsewhere in the document.
**Program numbers and FMIDs**

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

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

**Service and Maintenance URLs**

- z/OS Connect Enterprise Edition change history:  

- What’s new in z/OS Connect EE:  

**The z/OS Connect EE Knowledge Center URL**

- z/OS Connect EE Knowledge Center:  

**The WebSphere Application Server for z/OS Liberty Knowledge Center URL**

- z/OS Connect EE Knowledge Center:  
  [https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/as_ditamaps/welcome_liberty_zos.html](https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/as_ditamaps/welcome_liberty_zos.html)

**IBM developerWorks articles URL**

- Site of developerWorks articles related to z/OS Connect EE:  

**Additional IBM Support web pages**

- Details of the WebSphere Liberty Profile(WLP) upgrades shipped with z/OS Connect EE:  

- Techdocs – the Technical Sales Library  

- WLP Server configuration elements:  

**Broadcom Support web pages**

- Site of What ACF2 security setup is needed for IBM’s z/OS Connect Enterprise Edition V3.0?  

- Site of Setting up Liberty Server for z/OS with Top Secret  
  [https://knowledge.broadcom.com/external/article/37272/setting-up-liberty-server-for-zos-with-t.html](https://knowledge.broadcom.com/external/article/37272/setting-up-liberty-server-for-zos-with-t.html)
Installation and Initial Setup

Picture overview of the steps in this section

Notes:
1. SMP/E is used to install z/OS Connect EE using standard SMP/E installation processes. The result is a file system mounted at the location you specify and other SMP/E target data sets.
2. The zconsetup shell script must be executed to create a set of subdirectories under directory /var/zosconnect. This script must be run at least once per LPAR because one of these subdirectories is the z/OS Connect EE extensions directory. The extensions directory contains properties files that will be accessed by any z/OS Connect EE server started on this LPAR.
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 WebSphere Liberty Profile (Liberty) server with the z/OS Connect EE feature.
5. Copy the sample JCL procedures to your procedure library from the SBAQSAMP TLIB.
6. The angel process will be required in most circumstances. There may already be an angel active on your system. We will guide you through the process of configuring an angel specificity for z/OS Connect EE servers.
7. Install the z/OS Connect EE API Tool Kit on your workstation.
SMP/E install of z/OS Connect EE

IBM z/OS Connect EE (zCEE) is installed using standard SMP/E RECEIVE, APPLY and ACCEPT processes. This will require someone with SMP/E skills to accomplish this.

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


Follow the instructions in the program directory to install the product into its target OMVS filesystem and data sets.

The remainder of the section covers the details required to activate z/OS Connect on an LPAR and configure and start a z/OS Connect Liberty server.

Essential prerequisites

You will need the following:

- z/OS 2.2 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 2.2 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

Important: Before you continue with z/OS Connect configuration, a decision needs to be made regarding how OMVS ownership and permission bits will be set for the directories and files created in the next steps. Review the section entitled OMVS Ownership and Permission Bits Considerations to understand the significance of ownership and permission bits and decide which configuration solution is best for your environment.

OMVS Ownership/Permissions Bits Considerations

A common issue during the configuration of a z/OS Connect EE server is caused by the setting of ownership and permission bits of directories and files created during the customization process. Specifically, when configuration directories and files are created and configured by one identity when a different identity will be used to start the z/OS Server. This situation can easily result in the z/OS Connect server not having the required read/write access to the configuration directories and files and therefore not being able to properly initialize at all or have some key feature (e.g. SSL) disabled.

There are options for avoiding issues with ownership and permission bits. Two suggestions will be described in this section. Subsequent examples in this document will try to provide examples of using the options described in this section.

You do not have to use the techniques described here, but not using them or their equivalents will mean that configuring and managing servers may be more problematic.
Use of Surrogate Access

Since the identities associated with started task are normally restricted and cannot be used for accessing TSO or OMVS shells, one option to create the server’s configuration is to use RACF surrogate access. Surrogate access allows a designated administrative identity the ability to invoke commands and perform functions as if they were running under the identity that 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. In these examples, **LIBSERV** represents the identity under which the z/OS Connect server will be running and **adminUser** represent the administrative identity:

*Define a SURROGAT profile for the server’s SAF identity*

```bash
RDEFINE SURROGAT BPX.SRV.LIBSERV
```

*Define a SURROGAT profile to allow job submission as the server’s SAF identity*

```bash
RDEFINE SURROGAT LIBSERV.SUBMIT
```

*Permit an administrative identity to act as a surrogate of the Liberty task identity*

```bash
PERMIT BPX.SRV.LIBSERV CLASS(SURROGAT) ID(adminUser) ACC(READ)
PERMIT LIBSERV.SUBMIT CLASS(SURROGAT) ID(adminUser) ACC(READ)
```

*Refresh the SURROGAT in storage profiles*

```bash
SETROPTS RACLIST(SURROGAT) REFRESH
```

These commands allow the administrator identity (adminUser) to use the OMVS switch user command (su) with the -s flag (e.g. **su -s LIBSERV**) to switch identities to the Liberty’s started task identity (LIBSERV) and invoke OMVS commands (e.g. creating configuration directories and files) as the Liberty’s started task identity. This ensures all permission bits are set to the started task’s identity. Access to the SUBMIT resource allows an administrator identity (adminUser) to submit jobs as the Liberty’s servers task identity without the need to provide the password of started task user’s identity. This is done by simply adding **USER=LIBSERV** to the JOB card.

Permit access to Unix Privileges

An alternative to using a surrogate access is to permit the identity under which the customization will be done to enhanced Unix privileges. Specially, permitting the identity to Unix privileges SUPERUSER.FILESYS and SUPERUSER.FILESYS.CHOWN. See the z/OS Knowledge Center Using UNIXPRIV class profiles at URL https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.bpxb200/usspriv.htm

*Permit an administrative identity to write to any local directory or file*

```bash
PERMIT SUPERUSER.FILESYS CLASS(UNIXPRIV) ID(adminUser) ACC(CONTROL)
```

*Permit an administrative identity to change the ownership of any directory or file*

```bash
PERMIT SUPERUSER.FILESYS.CHOWN CLASS(UNIXPRIV) ID(adminUser) ACC(READ)
```

*Refresh the UNIXPRIV in storage profiles*

```bash
SETROPTS RACLIST(UNIXPRIV) REFRESH
```
Confirm that a Java runtime can be created.

The SAF identity used to perform z/OS Connect customization will need to be able run Java in OMVS. Confirm the SAF identity can execute Java by doing the following:

Add these export commands to the administrator .profile file in their home directory, e.g. /u/adminUser.

```bash
export JAVA_HOME= path_to_your_64-bit_Java_SDK
export $PATH= :.:$JAVA_HOME/bin:$PATH
```

**Tech Tip:** The .profile file can be found in the user’s home directory. Use ISHELL prior to entering OMVS or use the OMVS oedit command to change .profile and restart the OMVS command shell. Otherwise these environment variables will have to be exported each time an OMVS session is started.

You can use either an OMVS command (1) to confirm the Java environment can created or you can submit a job (2) for execution. Both methods are shown here.

1) Start an OMVS shell session using Telnet, SSH or the OMVS TSO command. Enter the commands below:

```bash
export JAVA_HOME= path_to_your_64-bit_Java_SDK
export $PATH= :.:$JAVA_HOME/bin:$PATH
java --version
```

**Tech Tip:** The OMVS in green above would not be required if the exports have been added to the user’s .profile.

You output should look something like this:

```
java --version
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 s90x-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
```

**N.B.:** In the above output, the build string of `build pmz6480sr3fp20-20161019_02(SR3 FP20)` indicates that the installed Java was built in October of 2016 as Service Release 3, fix pack 20. You should try to keep more current with Java service than a 4-year-old release of Java. The current level of Java for z/OS can be download from URL https://developer.ibm.com/javasdk/support/zos/ The string `s390x-64` confirms this is a 64-bit Java SDK.
2) Alternatively, the job below can also be used to check out the Java environment.

```plaintext
//****************************************************************
//*  Step IEBCOPY - Set common environment variables
//****************************************************************
//IEBGENER EXEC PGM=IEBGENER
//SYSPRINT DD SYSSOUT=*  
//SYSUT1  DD * 
JAVA_HOME=/usr/lpp/java/J8.0_64 
//SYSUT2  DD DSN=&&STDENV,DISP=(,PASS),
//   DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),SPACE=(TRK,(1,1)) 
//SYSSIN  DD DUMMY

//****************************************************************
//*  Step JAVA - Invoke the java -version command
//****************************************************************
//JAVA EXEC PGM=IKJEFT01,REGION=0M
//SYSERR   DD SYSOUT=* 
//STDOUT   DD SYSOUT=* 
//STDENV   DD DUMMY 
//SYSTSPRT DD SYSOUT=* 
//SYSTSIN  DD * 
BPXBATCH SH $JAVA_HOME/bin/java -version
```

**Tech Tip:** The IEBCOPY step (green) above would not be required if the exports have been added to the `.profile` file in the user’s home directory.

**N.B.:** If the `java` command fails (with an error JVMJ9VM011W EDC5204E) then it is likely because your identity does not can get the memory needed to create the JVM. Adjust³ 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))
```

³ 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.
Liberty angel Considerations

Access to privileged z/OS functions, e.g. SAF checks, writing SMF records, workload management (WLM), resource recovery services (RRS), etc. from or by a Liberty server are managed by the presence of an angel task and related security resource definitions. Some features of z/OS Connect EE will require that a Liberty angel be active 2.

If you have z/OSMF or other Liberty instances already running you may already have an angel active 3. Regardless, you should still use an angel configured to use the code provided with z/OS Connect.

N.B. If you do choose to use an existing angel process, it is probably not 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. Rather than upgrading the existing angel process, configuring another angel JCL started task procedure that references the WebSphere Liberty Profile (WLP) directories shipped with z/OS Connect and provide a unique name for that angel (e.g., NAME=angelName), for the z/OS Connect Liberty servers, see the next section.

Named angels

Each angel can be uniquely identified by a name at startup. An angel started with no name specified 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 (property com.ibm.ws.zos.core.angelName) then the default angel (i.e. the one with no name) will be selected. Another system property (com.ibm.ws.zos.core.angelRequired) can be set to require the successful connection to angel to continue the startup of the server. That is, if the required angel is not available, the Liberty server will shut itself down.

---

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

3 z/OSMF 2.1 is based on Liberty z/OS, and it requires the Angel for access to z/OS authorized services.
To provide these properties for a z/OS Connect EE server:

1. Create an options file for angel properties, e.g. `zcee.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
   ```

   Where `angelName` is the name of the angel to be used for security

2. Use the `JAVA_OPTIONS` environment variable in the z/OS Connect servers JCL and provide these properties in this file using the STDENV input. 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/zcee.options
   ```

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 section SAF SERVER and FACILITY profiles on page 19.

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 `LIBSERV` must be given READ access to this profile.

### Post SMP/E configuration steps

1. Create directory `/var/zosconnect` and mount a small ZFS filesystem at this mount point. This directory structure will be shared among all the z/OS Connect servers running on the same LPAR. The primary purpose of this directory structure is to provide a common location where properties (e.g. location of product executables, etc.) for some of the service providers not shipped with z/OS Connect can be located. This directory path is embedded in scripts, so it should not be changed. The contents of this directory are static and rarely will change. Mount points can be created in this directory structure and other filesystems mounted at these mount points.

   **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.

2. Create a mount point named `servers` in `/var/zosconnect` and mount a ZFS filesystem at this mount point. Ensure the identities that the identities under which the zCEE servers will run have write access to this directory.

   **Tech Tip:** The directory will be the default location for server configuration files and application artifacts. A ZFS filesystem mounted with AGGRGROW should be used to allow for growth.
3. Verify the product installation file system is mounted R/W. The `zconsetup` script 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 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/servers`.

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

4. Run the `zconsetup` script using either of the two options below. In either case, the identity under which the script executed must have the authority required to create a symbolic link in directory `/usr/lpp/IBM/zosconnect/v3r0/wlp/etc`

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

```
cd /usr/lpp/IBM/zosconnect/v3r0/bin
zconsetup install
```

**Tech Tip:** Including a period in the PATH environment variable back on page 12 implicitly adds the current directory to the PATH search order. Otherwise the `zconsetup` command would have had to be entered as `./zconsetup` or as `. zconsetup`.

The output should look like this:

```
zconsetup will use product feature directory /var/zosconnect/v3r0
zconsetup created the link to the product feature directory
zconsetup created /var/zosconnect/v3r0/extensions/zosconnect.properties
zconsetup created /var/zosconnect/v3r0/extensions/imsmobile.properties
zconsetup install completed successfully
```
b) As an alternative, submit the JCL below to execute this command in a batch job.

```jcl
//********************************************************************************
//*  STEP IEBCOPY - SET COMMON ENVIRONMENT VARIABLES
//********************************************************************************
//IEBGENER EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*  
//SYSUT1 DD *  
//JAVA_HOME=/usr/lpp/java/J8.0_64  
//ZCEEPATH=/usr/lpp/IBM/zosconnect/v3r0  
//SYSUT2 DD DSN=&&STDENV,DISP=(,PASS),  
//   DCB=(RECFM=TB,LRECL=80,BLKSIZE=80),SPACE=(TRK,(1,1))  
//SYSIN DD DUMMY
//********************************************************************************
//*  STEP ZCSETUP - INVOKE THE ZCONSETUP SCRIPT
//********************************************************************************
//ZCSETUP EXEC PGM=IKJEFT01,REGION=0M  
//SYSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
//STDENV DD DSN=&&STDENV,DISP=(OLD,DELETE)  
//SYSTSIN DD *  
//SYSTSPRT DD SYSOUT=*  
//BPXBATCH SH $ZCEEPATH/bin/zconsetup install
```  

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

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/bin</td>
<td>Product Code</td>
</tr>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/dev/</td>
<td>Java classes for user service providers</td>
</tr>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/doc</td>
<td>Java Doc zip file</td>
</tr>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/imsmobile</td>
<td>IMS Service Provider</td>
</tr>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/runtime/lib/</td>
<td>Feature Files</td>
</tr>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/wlp</td>
<td>WebSphere Liberty product code</td>
</tr>
<tr>
<td>/usr/lpp/IBM/zosconnect/v3r0/wlp/etc/extensions</td>
<td>Contains symbolic link to directory</td>
</tr>
<tr>
<td>/var/zosconnect/v3r0/zconnbt.zip</td>
<td>z/OS Connect EE build tool</td>
</tr>
<tr>
<td>/var/zosconnect/v3r0/extensions</td>
<td>Properties files for product features(^5)</td>
</tr>
<tr>
<td>/var/zosconnect/servers</td>
<td>Server configuration files and applications(^6)</td>
</tr>
</tbody>
</table>

\(^5\) 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.

\(^6\) The value for this directory is based on environment variable WLP_USER_DIR. The default value is shown.
SAF Resources

The SAF resources for z/OS Connect EE are best planned and created ahead of time.

SAF Groups and Server IDs

N.B. It is not required that the Liberty IDs be connected to a common group. Illustrated here is one approach. Identities LIBSERV, LIBANGL and group LIBGRP are just examples. Use the values appropriate for your system.

Note: For the initial setup we will keep things simple and use basic authentication and basic security. Basic authentication means a client with authentication using an identity and password combination. Basic security means the identity and passwords are maintain in the server’s configuration file, e.g. server.xml. To understand how to move beyond these simple security definitions, see Beyond the simple server.xml security elements on page 82. What follows are z/OS security elements that must be in place before operating the z/OS Connect EE server.

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 LIBGRP OMVS(AUTOGID) OWNER(SYS1)

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. This identity should be used for any z/OS Connect authentication/authorization purposes.

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

ADDUSER LIBANGL DFLTGRP(LIBGRP) OMVS(AUTOID

  HOME(/u/libangl) PROGRAM(/bin/sh)) NAME('Liberty angel')

  OWNER(LIBGRP) NOPASSWORD NOIDCARD

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

ADDUSER LIBSERV DFLTGRP(LIBGRP) OMVS(AUTOID

  HOME(/u/libserv) PROGRAM(/bin/sh)) NAME('Liberty Server')

  OWNER(LIBGRP) NOPASSWORD NOIDCARD
SAF STARTED profiles

SAF 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.

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(LIBANGL) GROUP(LIBGRP) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))

Creates the STARTED profile for the Liberty Profile server

RDEF STARTED serverProc.* UACC(NONE) STDATA(USER(LIBSERV) GROUP(LIBGRP) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))

Refreshes the STARTED class profiles

SETROPTS RACLIST(STARTED) REFRESH

SAF SERVER and FACILITY profiles

The SERVER and FACILITY profiles grant access to authorized services or features that the z/OS Connect EE may need to function. Some of these 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:

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(LIBSERV)

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

RDEF SERVER BBG.ANGEL.angelName8 UACC(NONE) OWNER(SYS1)
PERMIT BBG.ANGEL.angelName8 CLASS(SERVER) ACCESS(READ) ID(LIBSERV)

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(LIBSERV)

Controls which server processes can use BBGZSAFM for SAF authorization services

RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCREDS 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.
8 If so, then you do not need to create the profile, you need only grant your server ID READ to the profile.

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

 Controls which server processes can use BBGZSAFM for WLM services
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SOSWLM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.SOSWLM CLASS(SERVER) ACCESS(READ) ID(LIBSERV)

 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(LIBSERV)

 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(LIBSERV)

 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(LIBSERV)

 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(LIBSERV)

 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(LIBSERV)

 Controls access to EJBROLE definitions based on the SAF profile prefix in use for a server
RDEF SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(LIBSERV)

 Controls access to IFAUSAGE services (SMF)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(LIBSERV)

 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(LIBSERV)

 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(LIBSERV)

 Refreshes the SERVER and FACILITY 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.

---

9 BBG.SECPFX.BBGZDFLT is the default value for this resource. The security prefix is based on the profilePrefix value in the safCredentials configuration element in the server.xml.
• 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:


You can use either an OMVS command to create the server configuration or you can submit a job for execution. Both methods are shown here.

1) Start an OMVS shell session using Telnet, SSH or the OMVS TSO command.
   - Optionally, first switch to the ID you planned to use for the server’s started task using the OMVS `su` command, e.g. `su -s LIBSERV`
   - Go to the `/bin` directory where z/OS Connect EE is installed, e.g.

```
cd /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
```

- To create a server, use the `zosconnect` command:

```
zosconnect create serverName --template=templateName
```

Where `templateName` can be:
z/OS Connect EE Getting Started

- `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

**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 `apiRequester` feature. Additional 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.

Where `servername` is any value you wish, such as `zceesrvr`.

- Optionally, if you did not use the switch user command then the ownership of the configuration directory structure needs to be changed with a `chown` command

  `chown -R LIBSERV:LIBGRP SWLP_USER_DIR/servers/serverName`
2) To do the same from JCL, submit JCL like the JCL shown below:

```
//MYSERVER JOB 'ZCEE',CLASS=A,REGION=0M,MSGCLASS=H,NOTIFY=&SYSUID,USER=LIBSERV
.getOrElse(
    //******************************************************************************
    //* Step IEBCOPY - Set common environment variables
    //******************************************************************************
    IEBGENER EXEC PGM=IEBGENER
    //SYSPRINT DD SYSOUT=*      
    //SYSUT1 DD *
    JAVA_HOME=/usr/lpp/java/J8.0_64
    ZCEEPATH=/usr/lpp/IBM/zosconnect/v3r0
    SERVER=serverName
    TEMPLATE=zosconnect:default
    WLP_USER_DIR=/var/zosconnect
    USER=LIBSERV
    GROUP=LIBGRP
    //SYSUT2 DD DSN=&STDENV,DISP=(,PASS),
    //      DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),SPACE=(TRK,(1,1))
    //SYSIN DD DUMMY
    //******************************************************************************
    //* Step ZCEESRVR - Use the zosconnect command to create a server
    //******************************************************************************
    ZCEESRVR EXEC PGM=IKJEFT01,REGION=0M
    //SYSSERR DD SYOUT=*          
    //STDDOUT DD SYOUT=*         
    //STDENV DD DSN=&STDENV,DISP=(OLD,PASS)
    //SYSTSPRT DD SYOUT=*        
    //SYSTSIN DD *
    BPXBATCH SH $ZCEEPATH/bin/zosconnect create $SERVER +
    --template=$TEMPLATE
    //******************************************************************************
    //* Step CHOWN - Change directory and file ownership
    //******************************************************************************
    //CHOWN EXEC PGM=IKJEFT01,REGION=0M
    //SYSSERR DD SYOUT=*          
    //STDDOUT DD SYOUT=*         
    //STDENV DD DSN=&STDENV,DISP=(OLD,DELETE)
    //SYSTSPRT DD SYOUT=*        
    //SYSTSIN DD *
    BPXBATCH SH chown -R $USER:$GROUP $WLP_USER_DIR/servers/$SERVER
```

**Tech Tip:** If surrogate access has been enabled then the USER JCL parameter can be added to the JOB as shown above. In this example, the job will execute under the **LIBSERV** identity and all directories and files will be owned by **LIBSERV** with the appropriate permissions bits set. If surrogate access is not used, then the directory and file ownership can be changed to LIBSERV by executing the change owner command in step CHOWN. Either the USER parameter or the CHOWN step should be used.

After the server is created 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.
TCP ports and host element
A few minor updates to /var/zosconnect/servers/serverName/server.xml may be required at this point.

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.

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

• Save the file.

**Tech Tip:** Setting a port to minus 1 (-1) disables that protocol.
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 for your environment.

Do the following:

- Copy the sample server JCL from member BAQSTRT in your SMP/E SBAQSAMP target library to your PROCLIB, renaming it as you copy. 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.

  Tech Tip: Characters in columns 73-80 will cause havoc if they appear in the input to the STDENV DD statement.

- Rename the procedure so it matches the STARTED profile you created for the server.

- Customize the server JCL:

  ```
  /BAQSTRT  PROC  PARMS='servername'  
  /*
   *(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>'
   */
   /* ZCON   EXEC PGM=BPXBATS,REGION=0M,MEMLIMIT=8G,
   */ PARM='PGM &ZCONHOME./bin/zosconnect run &PARMS.'
   */ STDOUT DD SYSDUMP=*  
   */ STDERR DD SYSOUT=*  
   */ STDIN  DD DUMMY  
   */ STDENV DD *
   _BPX_SHAREAS=YES
   _CEE_RUNOPTS=HEAPPOLS(ON),HEAPPOLS64(ON)
   JAVA_HOME=<Java home directory>
   WLP_USER_DIR=<User directory>
   #JVM_OPTIONS=<Optional JVM parameters>
   */
   PEND
  ```

Notes:

1. Change `servername` to match the name of the server created earlier (case matters).
2. 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.
3. Set `JAVA_HOME` to the path to your 64-bit IBM Java SDK, e.g. `/usr/lpp/java/J8.0_64`

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

**Tech Tip:** The same value used for `WLP_USER_DIR` that was used when creating the server needs to be exported in the JCL used to start the server. `WLP_USER_DIR` is a Liberty environment variable. See Liberty Environment Variables on page 139 for information on other Liberty environment variables.

5. If you will be using a named angel this property is where you can specify the angel’s name, see the information on page 14 regarding named angels.

If you intend to use an already-existing angel process, then skip over the following steps. Otherwise, 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='',SAFLOG=N                      
     //****----------------------------------------------------------------- 
     //*  SET ROOT='/usr/lpp/IBM/zosconnect/v3r0/wlp'                       
     //****----------------------------------------------------------------- 
     //* 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=NOLIMIT,                    
     //  PARM='PGM &ROOT./lib/native/zos/s390x/bbgzangl COLD=&COLD NAME=X    
     //  &NAME &PARMS SAFLOG=&SAFLOG'                                     
     //STDOUT    DD SYSOUT=*                                              
     //STDERR    DD SYSOUT=*                                              
     //                                                                  
     //* =================================================================** 
```

**Notes:**

1. A name can be given to an angel either by providing a value in the NAME parameter in the JCL by overriding the `NAME` parameter when the angel is started, e.g.

   Start the angel with MVS command `S angelProc` 
   
   `S BAQZANGL,NAME=PRODUCTION`

2. We recommend the `SAFLOG` parameter be set to `Y`.

---

10 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.
3. Change the `SET ROOT=` value so it reflects the install location for z/OS Connect EE, including the `/wlp` sub-directory.

- Verify the angel received the authorization ID you intended, see message IEF695I. This validates the STARTED profile you created for the angel process.

**Tech Tip:** The SAFLOG parameters was added in a recent Liberty drop. If this parameter is set to `Y` additional security related messages will be written to the JES messages and console if a Liberty does not have authorization to use an angel control privilege function. For example, if a server is requesting access to SAF authentication and the necessary permits have not been done, this message will appear.

```
ICH408I USER(LIBSERV ) GROUP(LIBGRP ) NAME(LIBERTY SERVER
BBG.AUTHMOD.BBGZSAFM.SAFCRED CL(SERVER   )
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ   ) ACCESS ALLOWED(NONE   )
```

**Setup of basic security**

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.
Create a file named basicSecurity.xml in a directory, e.g. /var/zosconnect and add the configuration statements shown here.

```xml
<server description="basic security">
  <!-- Enable features -->
  <featureManager>
    <feature>appSecurity-2.0</feature>
  </featureManager>
  <keyStore id="defaultKeyStore" password="Liberty"/>
  <webAppSecurity allowFailOverToBasicAuth="true"/>
  <basicRegistry id="basic1" realm="zosConnect">
    <user name="Fred" password="fredpwd"/>
  </basicRegistry>
  <authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
      <user name="Fred"/>
    </security-role>
  </authorization-roles>
</server>
```

**Notes:**

1. Enables application security, which z/OS Connect EE will use. 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.

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.

- Add an include element the basicSecurity.xml file created above as shown below:

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

**Tech Tip:** For more information about including configuration elements from other files and sources see URL [https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_setup_includes.html](https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_setup_includes.html)

- Save the files.
For a more detailed description of the steps required to configure a basic z/OS Connect server see the security exercise *IBM z/OS Connect EE V3.0 Basic Configuration* at URL

https://tinyurl.com/yxw7c8ot

Next start the server and verify basic operations:

- Start the z/OS Connect server with the following command
  
  \[
  S \text{ serverProc,PARMS='serverName'}
  \]

  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, see message IEF695I. This validates the STARTED profile you created for the server.

- Go to the `/var/zosconnect/servers/serverName/logs` directory

  ![Notes](https://via.placeholder.com/150)

  **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.

- Look in the messages.log file. You should see the following messages\(^{12}\). See notes that follow:

---

\(^{12}\) The messages may occur in a slightly different order. That’s okay; the important thing is the various success indicators are present.
CWWKE0001I: The server serverName has been launched.
CWWKB0125I: This server requested a REGION size of 0KB. The below-the-line storage limit is 8MB and the
above-the-line storage limit is 1542MB.
CWWKB0126I: MEMLIMIT=1000. MEMLIMIT CONFIGURATION SOURCE=JCL.
CWWKB0124I: Angel ZCEE is required, server was configured to wait up to 0 seconds to connect to the
targeted Angel.
CWWKB0122I: This server is connected to the ZCEE angel process.
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.
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 PRODMEG is available.
CWWKB0104I: Authorized service group ZOSAI0 is available.
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.
CWWKB0121I: The server process UMASK value is set to 0000.
CWWKE0002I: The kernel started after 1.688 seconds
CWWKF0007I: Feature update started.
CWWKS0007I: The security service is starting...
DYNA1001I: WebSphere Dynamic Cache instance named baseCache initialized successfully.
DYNA1071I: The cache provider default is being used.
DYNA1056I: Dynamic Cache (object cache) initialized successfully.
CWWK1508I: Creating the SSL certificate. This may take a few seconds.
CWWKS113I: Creating the LTPA keys. This may take a few seconds.
CWWKS1123I: The collective authentication plugin with class name NullCollectiveAuthenticationPlugin has
been activated.
BAQR0000I: z/OS Connect Enterprise Edition version 3.0.34.0 (20200615-1601)
DYNA1056I: Dynamic Cache (object cache) initialized successfully.
DYNA1001I: WebSphere Dynamic Cache instance named zosconnect_safcache initialized successfully.
DYNA1071I: The cache provider default is being used.
CWWK00219I: TCP Channel defaultHttpEndpoint has been started and is now listening for requests on host *
(IPv4) port 9080.
CWWKS4104A: LTPA keys created in 1.629 seconds. LTPA key file:
/var/zosconnect/servers/serverName/resources/security/ltpa.keys
CWWKF0012I: The server installed the following features: [appSecurity-2.0, distributedMap-1.0,
jaxrsClient-2.0, jndi-1.0, json-1.0, servlet-3.1, ssl-1.0, zosconnect:zosConnect-2.0,
zosconnect:zosConnectCommands-1.0].
CWWKF0008I: Feature update completed in 4.548 seconds.
CWWKF0001I: The serverName server is ready to run a smarter planet. The serverName server started in 5.038
seconds.
CWWKS4105I: LTPA configuration is ready after 1.642 seconds.
SRVE0169I: Loading Web Module: z/OS Connect.
SRVE0250I: Web Module z/OS Connect has been bound to default_host.
SESN8501I: The session manager did not find a persistent storage location; HttpSession objects will be
stored in the local application server's memory.
SESN0176I: A new session context will be created for application key default_host/
SESN0172I: The session manager is using the Java default SecureRandom implementation for session ID
generation.
SRVE9103I: A configuration file for a web server plugin was automatically generated for this server at
/var/zosconnect/ser vers/serverName/logs/state/plugin-cfg.xml.
DYNA1056I: Dynamic Cache (object cache) initialized successfully.
successful.
CWWK9122I: For URL /* in application com.ibm.zosconnect, the following HTTP methods are uncovered, and
accessible: OPTIONS
CWPK10803A: SSL certificate created in 4.248 seconds. SSL key file:
/var/zosconnect/servers/serverName/logs/state/plugin-cfg.xml.
Successfully loaded default keystore: /var/zosconnect/servers/serverName/resources/security/key.p12 of
type: PKCS12
CWWK00219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now listening for requests on host *
(IPv4) port 9443.
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.
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.

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 not suitable for anything but testing purposes. If you are interested in seeing how to enable SAF to perform these security functions, see section Using SAF for controlling z/OS Connect EE access on page 87.

Use a browser and enter the following URL:

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

where:
• The protocol is https (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)

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 134 and follow the instructions later in this section for using cURL to verify the server.

• 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.13
• You should then get a basic authentication prompt:

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

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

- Optionally, verify the server using cURL. For details regarding this test tool see Testing z/OS Connect Services Using cURL on page 134.

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

---
14 It's really /P <jobname>, but earlier you started the server with just the proc name, so that becomes the jobname as well.
■ You should then see something like the following:

```
{"apis":[]}
```

Again, 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`\(^{15}\). 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.

**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.

---

\(^{15}\) 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.

IBM provides the z/OS Connect API Toolkit is provided as an Eclipse plug-in in two service streams. One service stream, called Aqua 3.1, is based on Eclipse 4.6 Neon and the second stream, called Aqua 3.2 is based on Eclipse 4.8 Photon. Aqua service streams are not interchangeable, e.g. you cannot install an z/OS Connect Toolkit plug-in for the Aqua 3.1 service stream in an Eclipse environment based on Eclipse 4.8 or later, and vice-a-versa. Always be aware of any existing Eclipse environment and if you are installing for the first time, use the most current level.

There are two steps to the installation process: (1) installing an Eclipse environment, and (2) installing the z/OS Connect EE API Toolkit plug-in into the Eclipse platform[16]. The Eclipse environment can be an Eclipse environment obtain directly from the eclipse.org web site or an Eclipse environment established by installation of an Eclipse base product like IBM Explorer for z/OS 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 or IBM Explorer for z/OS Aqua 3.2

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

Installing the z/OS Connect EE V3.0 API Toolkit in an Eclipse environment

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:

1) From your open Eclipse platform[17], select Help → Install New Software.
2) Then click the "Add" button:

[Image: Available Software window]

[17] Either IBM z/OS Explorer or from an Eclipse installation.
3) Provide a name (such as "z/OS Explorer Composite Update Site" and then for Location enter the URL appropriate for your level of Eclipse. For Neon enter the Aqua 3.1 site or for Photon enter the Aqua 3.2 site.


4) Click **OK**.

5) 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:

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

7) Click **Next** twice and then agree to the license agreement. Then click **Finish**.
8) When the installation is complete, there will be a message that you need to restart, click Yes to continue.

![Software Updates dialog box]

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

**Installing the z/OS Connect EE V3.0 API Toolkit in an IBM Installation Manager environment**

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


![IBM Explorer for z/OS Aqua]

**Tech Tip:** Clicking the **Host Components** button provides a path for obtaining the z/OS components for installing and activating the *IBM Explorer for z/OS* server. Activating this server enables the remote system explorer server-side daemon. This daemon provides access to z/OS resources (data sets, spool, etc.) to the IBM Explorer for z/OS client.
2) Before continuing, understand the differences between installation format options (Installation Manager versus Eclipse p2), see below. Either is acceptable but once a format is chosen, you are committed to applying future updates using the same format, again, these formats cannot be intermixed.

3) On the Product selection view, check the box beside IBM z/OS Connect EE. Then click the Select button.

4) On the Aqua version selection view, click the Select button beside your choice for Aqua version.

5) On the Installation client view, click the Select button beside your choice between IBM Installation Manager or Eclipse p2.

6) On the Starting point view, click the Select button from beside your choice for the starting point.

7) On the Download and install view, select the Download button and follow the instructions.

Eclipse will restart. When it is open, select Window → Open Perspective → Other and select z/OS Connect Enterprise Edition. Click OK to continue.
9) In the upper-right corner you should see something like this:

![Image]

**N.B.:** 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:

![Image]

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

10) 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:

  ![Workspace Launcher](image)

  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.

![Perspective Selection](image)
• To add a connection to the z/OS Connect Server select z/OS Connect Enterprise Edition connection in the Host connections 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, 
  
  9443 
  for the Port Number, 
  
  check the box for Secure connection (TLS/SSL) 
  and then click the Save and Connect button.

![Add z/OS Connect Enterprise Edition Connection](image1)

• 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.

![Server certificate alert – Accept this certificate](image2)

• The status icon beside wg31:9443 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.
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

The API and services accessed in this section were developed based on the contents of the exercise a for Developing RESTful APIs for CICS program at URL https://tinyurl.com/y5y9lqtl. The CICS container enabled application can be obtained from Github site https://tinyurl.com/y3goh839. The target CICS program is CSCVINC and the copy books for request and response containers are CSCCREQ and CSCCRESP. The program accesses the CICS IVP VSAM file, commonly called FILEA. To complete this exercise on your own system compile program CSCVINC using the copy books and install the program and the CICS IVP FILEA into a CICS region. Then use the same copy books in the tooling to crate the service archive files.

In this scenario the CICS region accessed 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

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 to enable IPIC.

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 or in an included file, see the notes below:

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

<zosconnect_cicsIpicConnection id="cscvinc" host="wg31.washington.ibm.com" port="1491"/>
```

Notes:

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

Tech Tip: For the other zosconnect_cicsIpicConnection options, e.g. connectionTimeout, sendSessions, etc. see URL https://tinyurl.com/vyquyday
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 TCPIService 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:

<table>
<thead>
<tr>
<th>TCPIPService resource attribute</th>
<th>Value required</th>
</tr>
</thead>
<tbody>
<tr>
<td>URM</td>
<td>DFHISAIP</td>
</tr>
<tr>
<td>Port Number</td>
<td>A numeric value of an available port, e.g. 1491</td>
</tr>
<tr>
<td>Status</td>
<td>OPEN</td>
</tr>
<tr>
<td>Protocol</td>
<td>IPIC</td>
</tr>
<tr>
<td>Transaction</td>
<td>CISS</td>
</tr>
</tbody>
</table>

**Tech Tip:** Some of the other key CICS TCPIService related attributes, e.g. sendSessions are configured in a corresponding CICS IPCONN resource definition. See URL
Developing RESTful Services for CICS

Once the IPIC configuration is completed Development of the API is not required to test the infrastructure.

This document shows how to develop and deploy CICS services as well as showing how to develop and deploy APIs that consume these services following the instructions in the exercise Developing RESTful APIs for CICS program document at URL https://tinyurl.com/y5y9lqtl. 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 a CICS program, you should have at least 4 services deployed to the server. These services are cscvincDeleteService, cscvincInsertService, cscvincSelectService and cscvincUpdateeService. 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 128 or section Testing z/OS Connect Services Using cURL on page 134 to test the 3 services.

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

```json
{
    "cscvincSelectServiceOperation": {
        "cscvincContainer": {
            "request": {
                "filea": {
                    "employeeNumber": "222222"
                }
            }
        }
    }
}
```
With expected JSON response message:

```json
{
   "cscvincSelectServiceOperationResponse": {
      "cscvincContainer": {
         "response": {
            "CEIBRESP": 0,
            "USERID": "CICSUSER",
            "files": {
               "date": "26 11 81",
               "amount": "$0022.00",
               "address": "FRANKFURT, GERMANY",
               "phoneNumber": "20034151",
               "name": "DR E. GRIFFITHS",
               "employeeNumber": "222222"
            },
            "CEIBRESP": 0
         }
      }
   }
}
```

- For service `cscvincInsertService`, use URL

```json
{
   "cscvincInsertServiceOperation": {
      "cscvincContainer": {
         "request": {
            "files": {
               "employeeNumber": "948487",
               "name": "Mitch Johnson",
               "address": "Cary NC",
               "phoneNumber": "0065",
               "date": "11/23/20",
               "amount": "000100"
            }
         }
      }
   }
}
```
With expected JSON response message:

```json
{
   "cscvincInsertServiceOperationResponse": {
      "cscvincContainer": {
         "response": {
            "CEIBRESP2": 0,
            "USERID": "CICSUSER",
            "CEIBRESP": 0
         }
      }
   }
}
```

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

```json
{
   "cscvincUpdateServiceOperation": {
      "cscvincContainer": {
         "request": {
            "filea": {
               "employeeNumber": "948487",
               "name": "Mitch Johnson",
               "address": "Cary NC",
               "phoneNumber": "0065",
               "date": "11/23/20",
               "amount": "000200"
            }
         }
      }
   }
}
```

With expected JSON response message:

```json
{
   "cscvincUpdateServiceOperationResponse": {
      "cscvincContainer": {
         "response": {
            "CEIBRESP2": 0,
            "USERID": "CICSUSER",
            "CEIBRESP": 0
         }
      }
   }
}
```
For service `cscvincDeleteService`, use URL

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

JSON request message:

```json
{
    "cscvincDeleteServiceOperation": {
        "cscvincContainer": {
            "request": {
                "filea": {
                    "employeeNumber": "948487"
                }
            }
        }
    }
}
```

With expected JSON response message:

```json
{
    "cscvincUpdateServiceOperationResponse": {
        "cscvincContainer": {
            "response": {
                "CEIBRESP2": 0,
                "USERID": "CICSUSER",
                "CEIBRESP": 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.

**Security and CICS**

For an example of the steps required to enable security between a CICS region and z/OS Connect, see the security exercise *IBM z/OS Connect EE V3.0 Customization – Security and CICS* at URL

https://tinyurl.com/y3ub4nav
IMS TM RESTful APIs

The API and services accessed in this section were developed based on the contents of the exercise a for Developing RESTful APIs for IMS Transaction at URL https://tinyurl.com/y5bopv2e. The IMS copy book, DFSIVTNO, used in this exercise can be obtained from Github site https://tinyurl.com/y3goh839.

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 21 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.

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 a 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:

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:

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\(^{18}\) that represents all the services that are auto-created with the IMS support:

```
{
  "zosConnectServices": [
    {
      "serviceName": "IMSProvidedIMSmobileService",
      "serviceDescription": "IBM-provided IMS mobile service that plugs the gateway server and an IMS Connect subsystem",
      "serviceProvider": "zosmobile",
      "serviceURL": "https://zosmobile.server.com:9463/zosConnect/services/IMSProvidedIMSmobileService"
    },
    {
      "serviceName": "IMSConnectionAdmin",
      "serviceDescription": "IBM-provided IMS mobile administrative service for managing IMS connections",
      "serviceProvider": "zosmobile",
      "serviceURL": "https://zosmobile.server.com:9463/zosConnect/services/IMSConnectionAdmin"
    },
    {
      "serviceName": "IMSMessageAdmin",
      "serviceDescription": "IBM-provided IMS mobile administrative service for managing IMS transaction messages",
      "serviceProvider": "zosmobile",
      "serviceURL": "https://zosmobile.server.com:9463/zosConnect/services/IMSMessageAdmin"
    },
    {
      "serviceName": "IMSTransAdmin",
      "serviceDescription": "IBM-provided IMS mobile administrative service for managing IMS transactions",
      "serviceProvider": "zosmobile",
      "serviceURL": "https://zosmobile.server.com:9463/zosConnect/services/IMSTransAdmin"
    },
    {
      "serviceName": "IMSPROPERTYsAdmin",
      "serviceDescription": "IBM-provided IMS mobile administrative service for managing IMS interaction properties",
      "serviceProvider": "zosmobile",
      "serviceURL": "https://zosmobile.server.com:9463/zosConnect/services/IMSPROPERTYsAdmin"
    },
    {
      "serviceName": "IMSServiceAdmin",
      "serviceDescription": "IBM-provided IMS mobile administrative service for managing mobile services",
      "serviceProvider": "zosmobile",
      "serviceURL": "https://zosmobile.server.com:9463/zosConnect/services/IMSServiceAdmin"
    }
  ]
}
```

\(^{18}\) 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.

**Note:** This test does not exercise a connection to IMS Connect. You will do that after you have configured a service and interaction definition.
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 **Send** 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 in an easy to read format, see below.

### Using cURL

```bash
curl -X POST --user Fred:fredpwd --insecure
```

---

**Tech-Tip:** In the above example:

```bash
--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**.

![z/OS UNIX Directory List](image)

- 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 `IVPI` (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 an IMS transaction

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://tinyurl.com/y5bopv2e. 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 at least 4 services deployed to the server. These services are ivtnoADDService, ivtnoDeleteService, ivtnoDisplayService and ivtnoUpdateService. The services can be used to test connectivity to IMS 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 128 or section Testing z/OS Connect Services Using cURL on page 134 to test the 3 services.


To display a phone book contact use JSON request message:

```json
{
    "phonebookRequest": {
        "lastName": "LAST1"
    }
}
```

With expected JSON response message:

```json
{
    "phonebookResponse": {
        "lastName": "LAST1",
        "firstName": "FIRST1",
        "zipCode": "D01/R01",
        "extension": "8-111-1111",
        "message": "ENTRY WAS DISPLAYED"
    }
}
```
For the `ivtnoDeleteService` service use URL

```plaintext
```

To delete a phone book contact use JSON request message:

```json
{
    "phonebookRequest": {
        "lastName": "LAST1"
    }
}
```

With expected JSON response message:

```json
{
    "phoneBookResponse": {
        "message": "ENTRY WAS DELETED"
    }
}
```

For the `ivtnoAddService` service use URL

```plaintext
```

To add a phone book contact use JSON request message:

```json
{
    "phonebookResponse": {
        "lastName": "LASTZ",
        "firstName": "FIRSTZ",
        "zipCode": "D0Z/R0Z",
        "extension": "8-111-5555"
    }
}
```

With expected JSON response message:

```json
{
    "phoneBookResponse": {
        "message": "ENTRY WAS ADDED"
    }
}
```

For the `ivtnoUpdateService` service use URL

```plaintext
```

To update a phone book contact use JSON request message:

```json
{
    "phonebookResponse": {
        "lastName": "LAST1",
        "firstName": "FIRST1",
        "zipCode": "D0Z/R0Z",
        "extension": "8-111-1111"
    }
}
```
With expected JSON response message:

```json
{
  "phoneBookResponse": {
    "message": "ENTRY WAS UPDATED"
  }
}
```

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.

**Security and IMS TM**

For an example of the steps required to enable security between an IMS transaction and z/OS Connect, see the security exercise *IBM z/OS Connect EE V3.0 Customization – Security when accessing an IMS Transactions* at URL [https://tinyurl.com/y5sykou4](https://tinyurl.com/y5sykou4)
IMS DB RESTful APIs

If your primary interest is accessing IMS database this section provides details on the steps required to activate the IMS database service provider.

Accessing an IMS database from a z/OS Connect EE (zCEE) server is done using ODBA through IMS Connect. In the example that will be shown in the section the IMS Connect task is running on TCP/IP host \texttt{wg31.washington.ibm.com} and listening on port 5555. The example shown here is based on the database used in the IMS Phone Book application (DFSIVD1)

Adding IMS Database support to a z/OS Connect server

Do the following:

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

```
<featureManager>
  <feature>zosconnect:dbService-1.0</feature>
</featureManager>

<ConnectionFactory id="DFSVIPAConn">
  <properties.imsdubJLocal databaseName="DFSIVPA"
   datastoreName="IVP1"
   datastoreServer="wg31.washington.ibm.com"
   driverType="4"
   portNumber="5555"
   user="USER1"
   password="USER1"
   flattenTables="True"/>
</ConnectionFactory>
```

Notes:

1. The feature that provides IMS database support.
2. This value must match the value that is specified for the \textit{connectionRef} property when a \textit{service} is developed in the API Toolkit.
3. The database name is the program specification block to be used to access the database.
4. The TCP/IP host name or IP address of the host on which the IMS Connect is listening for ODBA requests.
5. The port assigned to the DRDAPORT defined in the IMS Connect configuration, see below.

```
ODACCESS=(ODBMAUTOCONN=Y,IMSplex=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
   DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSAPPL)
```

- Save the file.
A Review of the IMS artifacts

PSB ATSIVPA was based on PSB DFSIVPA which is a sample program specification block provided with the IMS IVP. The sample PSB was modified to change the PROCOPT attribute from LS to A

```plaintext
ATSIVPA   PCB TYPE=DB, DBDNAME=IVPDB1, PROCOPT=LS, KEYLEN=10
          SENSEG NAME=A1111111, PARENT=0
          PSBGEN LANG=ASSEM, PSBNAME=DFSIVPA
END

The IVPDB1 data base description (DBD) referenced by PSB ATSSIVPA was based on also modified to by adding FIELD macros for FIRSTNME, PHONENBR and ZIPCODE.

```plaintext
DBD    NAME=IVPDB1, ACCESS=(HIDAM, OSAM)
DATASET DD1=DFSIVD1, DEVICE=3380, SIZE=2048
SEGM   NAME=A1111111, PARENT=0, BYTES=40, RULES=(LLV, LAST), x
       PTR=(TB, CTR)
FIELD   NAME=(A111111, SEQ, U), BYTES=010, START=00001, TYPE=C
FIELD   NAME=FIRSTNME, BYTES=010, START=00011, TYPE=C
FIELD   NAME=PHONENBR, BYTES=010, START=00021, TYPE=C
FIELD   NAME=ZIPCODE, BYTES=7, START=00031, TYPE=C
LCHILD NAME=(A1, IVPDB1I), POINTER=INDX, RULES=LAST
DBDGEN
FINISH
END
```
PSBGENs and DBDGENs were performed and utility DFS3UACB was used to update the IMS Catalog with these updates.

```plaintext
//ACBCAT EXEC PGM=DFS3UACB,REGION=OM
//STEPLIB DD DSN=DFSF10.SDFSRESL,DISP=SHR
//PROCLIB DD DSN=IMS15.PROCLIB,DISP=SHR
//DFSRESLB DD DSN=DFSF10.SDFSRESL,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSN DD SYSOUT=* SYSAEND DD SYSOUT=*
//IMS DD DSN=IMS15.PSBLIB,DISP=SHR
// PROCLIB DD DSN=IMS15.PROCLIB,DISP=SHR
// IMSACB DD DSN=IMS15.ACBLIB,DISP=OLD
// SYSSN DD UNIT=SYSDA,SPACE=(80,(100,100))
// SYSSN DD UNIT=SYSDA,SPACE=(256,(100,100)),DCB=KEYLEN=30
// SYSSN DD *
// BUILD PSB=DFSIVPA
// BUILD DBD=IVPDB1
// IMSACB01 DD DSN=*.IMSACB,DISP=OLD
// DFSVSAMP DD *
// VSRBF=32768,5
// VSRBF=8192,10
// IEFRDER DD DSN=JOHNSON.IMSLOG.P07,DISP=(DELETE),
//UNIT=SYSALLDA,SPACE=(CYL,(7)),
//DCB=(RECFM=VB,BLKSIZE=4096,
//LRECL=4092,BUFNO=2)
//DFS3PPRM DD *
//DLI,DFS3PU00,DFSCP001,.........,Y,N,..........,DFSDF=CAT
```

**Developing RESTful Services for an IMS database**

Once the IMS OTMA configuration is completed follow the instructions for the development and deployment of services in the *Developing RESTful APIs for IMS Database REST Services* document at URL [https://tinyurl.com/y3bw3hdy](https://tinyurl.com/y3bw3hdy). 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 an IMS Databases*, you should have services named `selectByName` and `selectByZipCode` deployed to the server. These services can be used to test connectivity to the IMS database from the z/OS Connect server. The service and infrastructure should be tested before developing APIs 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 128 or section Testing z/OS Connect Services Using cURL on page 134 to test the selectBYName service.

For the select by last name database service, use the URL below for the REST client as in
https://wg31.washington.ibm.com:9443/zosConnect/services/selectByByName?action=invoke

curl -X POST --user Fred:fredpwd --data @lastName.json --header "Content-Type: application/json" --insecure https://wg31:9443/zosConnect/services/selectByByName?action=invoke

To display a phone book contact by last name use JSON request message for lastName.json:

```json
{
  "request": {
    "A1111111" : "LAST1"
  }
}
```

With expected JSON response message:

```json
{"response":{"result":[{"A1111111":"LAST1","ZIPCODE":"D01\R01","FIRSTNME":"FIRST1","PHONENBR":"8-111-1111"}]}}
```

For the select by zip code service, use the URL below for the REST client as in

curl -X POST --user Fred:fredpwd --data @request.json --header "Content-Type: application/json" --insecure https://wg31:9443/zosConnect/services/selectByZipCode?action=invoke

To display a phone book contact by zip code use JSON request message for zipCode.json.

```json
{
  "request": {
    "ZIPCODE1": "D04/R04",
    "ZIPCODE2": "D05/R05"
  }
}
```

With expected JSON response message:

```json
{"response":{"result": [{"A1111111":"LAST4","ZIPCODE":"D04\R04","FIRSTNME":"FIRST4","PHONENBR":"8-111-4444"}, {"A1111111":"LAST5","ZIPCODE":"D05\R05","FIRSTNME":"FIRST5","PHONENBR":"8-111-5555"}]}}
```

**Security and IMS DB**

For an example of the steps required to enable security between an IMS transaction and z/OS Connect, see the security exercise IBM z/OS Connect EE V3.0 Customization – Security when accessing an IMS Database at URL https://tinyurl.com/yxp7v6ml
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).

```sql
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("zCEEService","selectEmployee")
The `selectEmployee` Db2 native REST service can be tested with Postman or `cURL` with URL `http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee` and JSON request message.

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

- **Using Postman**

  ![Postman](image)

- **Using `cURL`**

  ```bash
  curl -X POST --user USER1:USER1 --header "Content-Type: application/json"
  -d @selectEmployee.json

  "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 = :commission
  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 \texttt{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 \texttt{server.xml} and add the lines highlighted here in \textbf{bold} as shown, see the notes below:

\begin{verbatim}
<zosconnect_zosConnectServiceRestClientConnection id="db2conn"
    host="wg31.washington.ibm.com"
    port="2446"
    basicAuthRef="dsn2Auth" />

<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
    userName="USER1"
    password="USER1"/>
\end{verbatim}

Notes:
1. This value must match the value that is specified for the \texttt{connectionRef} property when a \texttt{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.

\textbf{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://tinyurl.com/y2t3rh4u. 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.

**Db2 Stored Procedure Considerations**

Accessing a Db2 stored procedure from a Db2 REST service has special considerations regarding the response message. When a Db2 REST services is created, the JSON request and response schemas are derived from "describe" information which is created and stored within the actual Db2 REST service package during the service creation process.

When a Db2 REST service is created that calls a stored procedure, the only “describe” information available is the name and signature of the stored procedure the maximum number of returned dynamic results sets (see DYNAMIC RESULTS SETS at URL https://tinyurl.com/yycwnban).

This means that for a Db2 REST service that calls a stored procedure, the JSON response schema for the results set does not have details on the contents of the results set entries, see below.

```json
"ResponseSchema": {
  "Schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "Resultset 1 Output": {
      "description": "Stored Procedure ResultSet 1 Data",
      "type": "array",
      "items": {
        "description": "ResultSet Row",
        "type": "object"
      }
    },
    "Anonymous ResultSets": {
      "type": "integer",
      "multipleOf": 1,
      "minimum": 0,
      "maximum": 1,
      "description": "Number of Anonymous ResultSets"
    },
    "StatusDescription": {
      "type": "string",
      "description": "Service Invocation status description"
    },
    "StatusCode": {
      "type": "integer",
      "multipleOf": 1,
      "minimum": 100,
      "maximum": 688,
      "description": "Service Invocation HTTP status code"
    }
  }
}
```

The results set entries are JSON arrays, where an individual array entry is the information returned by the stored procedure in JSON format.

The z/OS Connect API Editor given this response will only show the results as an array with no details of the array entries.

---

Db2 RESTful APIs
© 2016, 2020 IBM Corporation
Version Date: Tuesday, August 18, 2020
Let’s see how these details can be obtained by a client.

Below is an actual response message from invoking a Db2 REST services. Note that JSON properties in the results set array, e.g. `firstName`, `lastName`, `middleInitial`, `phoneNumber`, `department` and `employeeNumber`, do not appear in the JSON response schema for the reason noted above.

```json
{
  "Output Parameters": {},
  "StatusDescription": "Execution Successful",
  "ResultSet 1 Output": [
    {
      "firstName": "SALLY",
      "lastName": "KOANA",
      "middleInitial": "A",
      "phoneNumber": "4738",
      "department": "COI",
      "employeeNumber": "900038"
    },
    {
      "firstName": "DOLORES",
      "lastName": "QUINTANA",
      "middleInitial": "M",
      "phoneNumber": "4578",
      "department": "COI",
      "employeeNumber": "900138"
    },
    {
      "firstName": "HEATHER",
      "lastName": "NICOOLS",
      "middleInitial": "A",
      "phoneNumber": "1793",
      "department": "COI",
      "employeeNumber": "900148"
    },
    {
      "firstName": "KIM",
      "lastName": "NATZ",
      "middleInitial": "H",
      "phoneNumber": "1793",
      "department": "COI",
      "employeeNumber": "200148"
    }
  ]
}
```
These fields will have to be manually extracted from the result set property in the response message by the client. The sample code below shows how to do this in Java.

```
try {
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
    }

    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String output;
    StringBuilder stringBuffer = new StringBuilder();
    while ((output = bufferedReader.readLine()) != null) {
        stringBuffer.append(output);
    }
    JSONObject json = new JSONObject(stringBuffer.toString());
    JSONArray jsonarray = json.getJSONArray("resultset 1 output");
    JSONObject jsonobject = new JSONObject();
    for (int index = 0; index < jsonarray.length(); index++) {
        jsonobject = jsonarray.getJSONObject(index);
        if (jsonobject.has("employeeNumber")) {
            System.out.println("Employee Number : " + jsonobject.getString("employeeNumber"));
        }
        if (jsonobject.has("firstName")) {
            System.out.println("First Name : " + jsonobject.getString("firstName"));
        }
        if (jsonobject.has("lastName")) {
            System.out.println("Last Name : " + jsonobject.getString("lastName"));
        }
        if (jsonobject.has("middleInitial")) {
            System.out.println("Middle Initial : " + jsonobject.getString("middleInitial"));
        }
        if (jsonobject.has("phoneNumber")) {
            System.out.println("Phone Number : " + jsonobject.getString("phoneNumber"));
        }
        if (jsonobject.has("department")) {
            System.out.println("Department : " + jsonobject.getString("department"));
        }
        if (jsonobject.has("missingField")) { // This check to see if the field is present
            System.out.println("Missing Fields : " + jsonobject.getString("missingField"));
        } else {
            System.out.println("Field not present in JSON response");
        }
    }
}
```

The results are shown below:

![Test the Db2 RESTful APIs](image)

© 2016, 2020 IBM Corporation - 74 - Version Date: Tuesday, August 18, 2020
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 128 or section Testing z/OS Connect Services Using cURL on page 134 to test the Db2 services.

- For service selectEmployee use URL
  and use JSON request message:

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

With expected JSON response message:

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

- For service deleteEmployee use URL
  and use JSON request message:

```json
{
   "employeeNumber": "000020"
}
```
With expected JSON response message:

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

- For service `selectByRole` use URL
  

  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


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


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 \textit{displayEmployee} use URL 
\url{https://wg31.washington.ibm.com:9443/zosConnect/services/displayEmployee?action=invoke}
and use JSON request message:

\begin{verbatim}
{
    "employeeNumber": "948478"
}
\end{verbatim}

With expected JSON response message:

\begin{verbatim}
{
    "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
}
\end{verbatim}

\section*{Security and Db2}

For an example of the steps required to enable security between a z/OS Connect server and Db2, see the security exercise IBM \textit{z/OS Connect EE V3.0 Customization – Security and Db2} at URL 
\url{https://tinyurl.com/y2ukonfu}
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 the IBM 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.
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](https://tinyurl.com/y5xwgpw2) document at URL https://tinyurl.com/y5xwgpw2. 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](https://tinyurl.com/y5xwgpw2), you should have a service named `mqPutService` 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](https://tinyurl.com/y5xwgpw2) on page 128 or section [Testing z/OS Connect Services Using cURL](https://tinyurl.com/y5xwgpw2) on page 134 to test the `FileaQueue` service.

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

```json
{
    "MQPUTOperation": {
        "mqmessage": {
            "stat": " "
            "numb": "112354",
            "name": "Mitch",
            "addr": "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.

**Security and MQ**

For an example of the steps required to enable security between a z/OS Connect server and MQ, see the security exercise *IBM z/OS Connect EE V3.0 Customization – Security and MQ* at URL [https://tinyurl.com/yymum26t](https://tinyurl.com/yymum26t)
Security Topics

Beyond the simple server.xml security elements

Turning off SSL and/or 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 user authentication. Earlier in this document you 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 (HTTPS) or authentication. z/OS Connect EE provides a way to turn off either or both.

Requiring HTTPS and authentication are controlled by two configuration attributes for controlling security, but of which default to "true". Attribute requireSecure controls whether a connection to a z/OS Connect server must be on a HTTPS connection (requireSecure="true") or whether HTTP is supported (requireSecure="false"). Attribute requireAuth controls where an authenticated identity is required and whether this authenticated identity is used for subsequent authorization checks (requireAuth="true").

Turning off security at the Global Level

Both requireSecure and requireAuth default to true. Either one or both can be disabled for the entire server in the zosconnect_zosConnectManager configuration for the entire sever (globally) as shown below.

```
<zosconnect_zosConnectManager
    requireAuth="false"  requireSecure="false"/>
```

Turning off security at the API level

Security for a specific API can be controlled by adding a zosconnect_zosConnectAPIs configuration element and then adding explicit zosConnectAPI sub element for that API in a as shown below.

```
<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 either 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 security at the service level**

Security for a specific service can be controlled by adding an explicit `service` element for that Service in a `zosconnect_services` configuration element.

```
<zosconnect_services>
  <service name="inquireSingleService"
         requireAuth="false" requireSecure="false"/>
</zosconnect_service>
```

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.

---

This means that any security settings at the service level are only applicable when the service is invoked directory outside of an API, for an example of this see section *Test the Services* on page 45.

**Turning off security at the API requester level**

Security for a specific API requester can be controlled by adding an explicit `apiRequester` for that API requester in a `zosconnect_apiRequesters` configuration element.

```
<zosconnect_apiRequesters>
  <apiRequester name="ccscvincapi_1.0.0"
                requireAuth="false" requireSecure="false" />
</zosconnect_apiRequesters>
```

Requiring the authentication and authorization for use of an API requester can controlled for all API requester artifacts by using the `requireAuth` attribute on an `zosconnect_apiRequesters` configuration element or for a specific API requester using an individual `apiRequester` element. The use of HTTP or HTTPS can be controlled for a specific API requester in a an `apiRequester` element using the `requireSecure` attribute.

**Turning off security at the service endpoint level**

Security for a specific service, e.g. WOLA, DVM or services that use the service provider supplied by IBM MQ can be controlled by adding an explicit `zosconnect_zosConnectService` element.

```
<zosconnect_zosConnectService id="zosConnectDvsService"
                             invokeURI="/dvs" serviceDescription=""
                             serviceRef="dvsService" serviceName="dvsService"
                             requireAuth="false" requireSecure="false"/>
```

These services are generally services created using the BAQLS2JS utility or third-party providers.
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://tinyurl.com/y23ys392.

- 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
ADDSERVER WSGUEST RESTRICTED DFLTGRP(WSGUESTG) OMVS(AUTOID -
  HOME(/u/wsguest) PROGRAM(/bin/sh)) NAME('UNAUTHENTICATED USER') -
  NOPASSWORD NOOIDCARD

ADDSERVER FRED DFLTGRP(ZCEEUSRS) OMVS(AUTOID HOME(/u/fred/) -
  PROGRAM(/bin/sh)) NAME('USER FRED')
ALTUSER FRED PASSWORD(FRED) NOEXPIRE

RDEFINE APPL BBGZDFLT UACC(NONE) OWNER(SYS1) 4
PERMIT BBGZDFLT CLASS(APPL) RESET
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(WSGUEST,ZCEEUSRS) 5
SETR 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 `safSecurity.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 `basicSecurity.xml` file in the main `server.xml` file. SAF security can be enabled by creating an `safSecurity.xml` file and replacing the include `basicSecurity.xml` to an include of `safSecurity.xml`.

```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>
    </featureManager>
    <keyStore id="defaultKeyStore" password="Liberty"/>
    <webAppSecurity allowFailOverToBasicAuth="true"/>
    <safRegistry id="saf"/>
    <safAuthorization racRouteLog="ASIS"/>
    <safCredentials unauthenticatedUser="WSGUEST"
        profilePrefix="BBGZDFLT"/>
</server>
```

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,REFRESH,CONFIG
```

- 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:

```json
{
  "apis": [
    {
      "name": "catalog",
      "version": "1.0.0",
      "description": "",
    }
  ]
}
```

• Close the browser again and restart it and access the same URL. This time enter another identity, 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
The steps required to enable group checking for authorization will be shown in this section.

In this example, identity FRED will have administrative authority and USER1 will only have API execution authority. For more details on this topic, see the exercise zCEE Customization Basic Security which can be found at URL https://tinyurl.com/y23ys392.

- Add two new groups will 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)

Tech Tip: The z/OS Connect server is executing as an OMVS process. These means that all user identities and groups that used for security must have an OMVS segment. OMVS segments can be display by either an LU identity OMVS or LG group OMVS TSO command. Also note that each identity’s OMVS segment requires a valid HOME directory that actually does exist and one that the identity has R/W access.
Below is an example server.xml showing the configuration attributes that can be used to control access to z/OS Connect resources using group access checking. In this example the groups created above are shown along with other groups (lower case names) just to show the variety of combinations can be configured.

```
<zosconnect_zosConnectManager
  globalInterceptorsRef="interceptorList_g"
  globalAdminGroup="GMADMIN,admgrp1,admgrp2"
  globalOperationsGroup="GMOPERS,oprgp1,oprgp2"
  globalInvokeGroup="GMINVOKE,invgrp1,invgrp2"
  globalReaderGroup="GMREADR,readgrp1,readgrp2"/>

<zosconnect_authorizationInterceptor id="auth"/>
<zosconnect_authorizationInterceptor id="audit"/>
<zosconnect_zosConnectInterceptors id="interceptorList_g"
  interceptorRef="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_a"
  interceptorRef="auth,audit"/>

<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="catalog"
    runGlobalInterceptorsRef="true"
    adminGroup="GMADMIN,aapigrp1,aapigrp2"
    operationsGroup="GMOPERS,oapigrp1,oapigrp2"
    invokeGroup="GMINVOKE,iapigrp1,oapigrp2"
    readerGroup="GMREADR,rapigrp1,rapigrp2"/>
</zosconnect_zosConnectAPIs>

<zosconnect_apiRequesters>
  <apiRequester name="cscvincapi_1.0.0"
    runGlobalInterceptorsRef="false"
    interceptorsRef="interceptorList_a"
    adminGroup="GMADMIN,aaprgrp1,aaprgrp2"
    operationsGroup="GMOPERS,oaprgrp1,oaprgrp2"
    invokeGroup="GMINVOKE,iaprgrp1,oaprgrp2"
    readerGroup="GMREADR,raprgrp1,raprgrp2"/>
</zosconnect_apiRequesters>

<zosconnect_services>
  <service id="selectByEmployee" name="selectEmployee"
    runGlobalInterceptorsRef="false"
    interceptorsRef="interceptorList_a"
    adminGroup="GMADMIN,asrvgrp1,asrvgrp2"
    operationsGroup="GMOPERS,osrvgrp1,osrvgrp2"
    invokeGroup="GMINVOKE,isrvgrp1,isrvgrp2"
    readerGroup="GMREADR,rsrvgrp1,rsrvgrp2"/>
</zosconnect_services>
```

Note:

- If the `runGlobalInterceptorsRef` is set to “false” then the `interceptorRef` attribute must be supplied for group membership to be used for authorization.
- The use of a specific group control is optional as well as listing multiple groups.
• 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:

![Screenshot of Firefox showing the authentication window with Fred entered]

Fred in the administrator’s 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:

![Screenshot of Firefox showing an error message]


erroxDetails: "BAQR0409W: User user1 is not authorized to perform the request."
Next try to invoke an API.

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


- You should see the response below:

```json
{"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}},"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

• You should see the response below:

```json
{"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":"BAQR0409W: 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 in 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 exercise *zCEE Customization Basic Security* which can be found at URL https://tinyurl.com/y23ys392.

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

```rxml
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty') -
  OU('LIBERTY')) WITHLABEL('Liberty CA') TRUST -
  SIZE(2048) NOTAFTER(DATE(2022/12/31))
RACDCERT CERTAUTH EXPORT(LABEL('Liberty CA')) -
  DSN('USER1.CERTAUTH.CRT') FORMAT(CERTDER)
RACDCERT ID(LIBSERV) GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com') -
  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)
RACDCERT ID(LIBSERV) CONNECT(ID(LIBSERV) -
  LABEL('Liberty Client Cert') RING(Liberty.KeyRing)DEFAULT)
RACDCERT ID(LIBSERV) CONNECT(CERTAUTH LABEL('Liberty CA') -
  RING(Liberty.KeyRing))
PERMIT IRR.DIGTCERT.LISTRING -
  CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST -
  CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)
SETR RACLIST(FACILITY) REFRESH
```

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.

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

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

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:

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

<sslDefault sslRef="DefaultSSLSettings" />
<ssl id="DefaultSSLSettings"
  keyStoreRef="CellDefaultKeyStore"
  trustStoreRef="CellDefaultKeyStore" />
<keyStore id="CellDefaultKeyStore"
  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 identify 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\(^1\).
- Start Firefox and issue the following URL:

```
```

\(^{1}\) 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:

```
{
  "apis": [
    {
      "name": "catalog",
      "version": "1.0.0",
      "description": "",
      "adminUrl": "https://w31.washington.ibm.com:9443/zosConnect/apis/catalog"
    }
  ]
}
```

For more examples of using mutual authentication see the exercise **zCEE Customization Basic Security** at URL [https://tinyurl.com/y23ys392](https://tinyurl.com/y23ys392)

**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 server 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 certificate 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. 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://tinyurl.com/y23ys392.

- Stop the the z/OS Connect server.

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

```xml
<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 a Firefox web browser. That's next. For other browsers, e.g. Chrome or Edge, use the certmgr command to import the certificates into the Windows trust store.
• In Firefox, click on the to the **Open Menu** (1) icon and select the **Options** (2) tool.

![Firefox Options Menu](image1)

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

![Firefox Certificates Tab](image2)
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:

![Password Required dialog](image)

Enter the value 20 **secret** and click **OK**. You should see confirmation:

![Alert](image)

- 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**

---

20 The password specified when the certificate was exported.
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:

```
{
  "zosConnectServices": [
    {
      "ServiceName": "inquireSingle",
      "ServiceDescription": "",
      "ServiceProvider": "CICS-1.0",
    }
  ]
}
```

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

```
```
• You should see the response below:

```json
{"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.","errorDetails":"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.

```
```

• You should see the response below:

```json
```

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

For more examples of using mutual authentication see the exercise zCEE Customization Basic Security at URL https://tinyurl.com/y23ys392
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 with 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 certificates 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 `clientAuthentication` was enabled.

### CICS Identity Propagation

To enable the propagation of the authenticated identity onto CICS for CICS authorization checks make 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](https://tinyurl.com/y4d4bvop).

- 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.
IMS TM PassTickets

z/OS Connect service level V3.0.33 added support for the use of PassTickets between a z/OS Connect server and IMS Connect. This required some additional RACF resources which will be documented in this section.

- The PTKTDATA class was activated with a SETROPTS commands

```bash
SETROPTS CLASSACT(PTKTDATA) RACLIST(PTKTDATA)
SETROPTS GENERIC(PTKTDATA)
```

- PTKTDATA resources were defined for the target IMS Connect:

```bash
RDEFINE PTKTDATA IMSAPPL SSIGNON(KEYMASK(123456789ABCDEF0)) +
APPLDATA('NO REPLAY PROTECTION')
```

**Tech-Tip:** The value IMSAPPL was derived from the APPL name in the IMS Connect DRDACPORT definition, e.g.

```bash
DATASTORE=(GROUP=OTMAGRP,ID=IVP1, MEMBER=HWSMEM,DRU=HWSYDRU0,
TMEMBER=OTMAMEM, APPL=IMSAPPL)
```

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:

```bash
RDEFINE PTKTDATA IRRPTAUTH.IMSAPPL.* UACC(NONE)
PERMIT IRRPTAUTH.IMSAPPL.* ID(LIBSERV) CLASS(PTKTDATA) ACC(UPDATE)
```

- The RACF in storage profile need were updated:

```bash
SETROPTS RACLIST(PTKTDATA) REFRESH)
```

- The `ims-connections` configuration file needs to be updated by adding an `applicationName` attribute to `properties.gmoa` with a value that matches the APPL name configured in IMS Connect.

```xml
<server>
<imsmobile_imsConnection comment="" connectionFactoryRef="IVP1"
 id="IMSCONN"/>
<connectionFactory id="IVP1">
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000"
 applicationName="IMSAPPL" />
</connectionFactory>
</server>
```
IMS DB PassTickets

z/OS Connect service level V3.0.33 added support for the use of PassTickets between a z/OS Connect server and IMS Connect. 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)
  ```

- PTKTDATA resources were defined for the target IMS Connect:
  
  ```
  RDEFINE PTKTDATA IMSAPPL SSIGNON(KEYMASK(123456789ABCDEF0)) + APPLDATA('NO REPLAY PROTECTION')
  ```

  **Tech-Tip:** The value IMSAPPL was derived from the APPL name in the IMS Connect DRDAPORT definition, e.g.

  ```
  ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1), DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSAPPL)
  ```

  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.IMSAPPL.* UACC(NONE) PERMIT IRRPTAUTH.IMSAPPL.* ID(LIBSERV) CLASS(PTKTDATA) ACC(UPDATE)
  ```

- The RACF in storage profile need were updated:

  ```
  SETROPTS RACLIST(PTKTDATA) REFRESH)
  ```

- The server’s xml `properties.imsudbJLocal` element in the `connectionFactory` element was updated to add attribute `applicationName` with a value that match the APPL name defined in IMS Connect.

  ```
  <connectionFactory id="DFSIVPAConn">
  <properties.imsudbJLocal
databaseName="DFSIVPA"
datastoreName="IVP1"
datastoreServer="wg31.washington.ibm.com"
driverType="4"
portNumber="5555"
  applicationName="IMSAPPL"
flattenTables="True"/>
</connectionFactory>
```
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’)

<table>
<thead>
<tr>
<th>Tech-Tip:</th>
<th>The value DSN2APPL was derived from the Db2 LU name in the DSNL004I startup message, for example.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNL004I</td>
<td>-DSN2 DDF START COMPLETE 906</td>
</tr>
<tr>
<td>LOCATION</td>
<td>DSN2LOC</td>
</tr>
<tr>
<td>LU</td>
<td>USIBMWZ.DSN2APPL</td>
</tr>
<tr>
<td>GENERICLU</td>
<td>-NONE</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>WG31.WASHINGTON.IBM.COM</td>
</tr>
<tr>
<td>TCPPORT</td>
<td>2446</td>
</tr>
<tr>
<td>SECPORT</td>
<td>2445</td>
</tr>
<tr>
<td>RESPORT</td>
<td>2447</td>
</tr>
<tr>
<td>IPNAME</td>
<td>-NONE</td>
</tr>
<tr>
<td>OPTIONS:</td>
<td></td>
</tr>
<tr>
<td>PKGREL</td>
<td>= COMMIT</td>
</tr>
</tbody>
</table>

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(LIBSERV) CLASS(PTKTDATA) 
  ACCESS(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.

```xml
<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:

  ```text
  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 RACLST(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](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


  ```json
  {
    "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"."*"
```

**MQ services security**

To assert an identity from z/OS Connect to MQ for subsequent MQ authorization is done using JMS property `useCallerPrincipal`. This property must be specified on a service level in a `zosconnect_services` configuration element (see below). When properly `useCallerPrincipal` is set to true the JMS provider in z/OS Connect will send the z/OS Connect authorization identity to MQ.

```xml
<zosconnect_services>
  <service name="mqPutService">
    <property name="useCallerPrincipal" value="true"/>
  </service>
</zosconnect_services>
```

**MQ TLS security**

To enable TLS encryption and between a z/OS Connect server and a MQ queue manager the `jmsConnectionFactory` configuration element must specify a `transportType` of `CLIENT` and a channel which has an SSL cipher enabled.

For example, the `jmsConnectionFactory` connects to queue manager ZMQ1 over the network using channel `LIBERTY.SSL.SVRCONN`.

```xml
<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf" connectionManagerRef="ConMgr1">
  <properties.wmqJMS transportType="CLIENT"
    queueManager="ZMQ1"
    channel="LIBERTY.SSL.SVRCONN"
    hostName="wg31.washington.ibm.com"
    sslcipherSuite="SSL_RSA_WITH_AES_256_CBC_SHA256"
    port="1422"/>
</jmsConnectionFactory>
```
The `LIBERTY.SSL.SVRCONN` channel has been defined to the queue manager using these commands:

```
DEFINE CHANNEL ('LIBERTY.SSL.SVRCONN') REPLACE +
   CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   SSLCIPH('TLS_RSA_WITH_AES_256_CBC_SHA256')
SET CHLAUTH('LIBERTY.SSL.SVRCONN') ACTION(REPLACE) +
   TYPE(SSLPEERMAP) ADDRESS('**') CHCKCLNT(ASQMGR) +
   SSLCERTI('CN=MQ CA,OU=ATS,O=IBM') SSLPEER('OU=ATS') USERSRC(CHANNEL)
```

N.B.: The channel authentication record simply validates the certificate was issued by the specific certificate authority.

To relate the JSSE ciphers specified in the JMS connection factory to the corresponding IBM or Oracle JRE MQ Cipher Suite names use this URL:

**z/OS Connect and AT-TLS**

In some situations, z/OS Connect requires Application Transparent – TLS (AT-TLS) to enable encryption and security between a z/OS Connect server from an inbound REST client or between a z/OS Connecter server to an outbound endpoint. 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 (client or server) need not be aware that network traffic is being encrypted and/or digital certificates are being used for security.

AT-TLS is needed when TLS is required for communications with Db2 and IMS Connect and/or when TLS is required by an API client requester application running in MVS batch job or an IMS region. This section will describe configuring AT-TLS for an API requester application running in other non-CICS environments. For details and examples of configuring AT-TLS in other scenarios, see the security exercises at URL [https://tinyurl.com/y3uwq9nl](https://tinyurl.com/y3uwq9nl)

**HTTPS Communication Options**

The diagram below shows the flows for inbound communication to a z/OS Connect server. REST Clients such as browsers, Postman and 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. An inbound policy provides the TLS server functions for inbound HTTPS request connecting when connecting to an HTTP port and an outbound policy provides the TLS client functions for outbound request originating from a non-TLS enabled client.
AT-TLS policies are stored in a file in an OMVS directory which contains configuration information 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 a highly 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 from the Configuration Assistant used to configure a simple configuration. Not all the steps for using the Configuration Assistant will be shown, just the key screens. For details see the site referenced earlier.

**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 (\textit{Libery.KeyRing}) created in \textit{Using RACF for TLS and trust/key store management} on page 91.

\textbf{Tech-Tip:} The same key ring can be referenced in a AT-TLS policy, or in Liberty server’s configuration file and a CICS region’s resource definitions.

The server’s end points are configured as shown below:

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

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

\textbf{HTTPS Client Traffic Descriptor}

As shown below, the HTTPS client traffic descriptor defines the remote port for the server as being HTTPS port 9173 and the client coming from any local port. 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 specific \textit{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.
The remote port is 9173 which is configured in the server as an JSSE HTTPS port. This is a AT-TLS client interacting with a JSSE port.

**HTTP Client Traffic Descriptor**

The *Configuration Assistant* screen below identifies the target port and handshake role in a *Traffic Descriptor* outbound policy for the HTTP port 9120. As shown below this descriptor identifies the remote port for the server as being 9120 “outbound” from any local HTTP client request. 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 handshake.

![Configuration Assistant Screen](image)

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) for connection request to port 9120. That is the next example.
Server Traffic Descriptor (AT-TLS server and TLS client)

This outbound AT-TLS policy identifies the local port and handshake role in a Traffic Descriptor inbound policy to HTTP port 9120. As shown below this descriptor identifies the local port for the server as being 9120 from any local port. 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 configuration with an MVS modify command, 

F PAGENT,UPDATE

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

When an API requester uses the options below while running under identity JOHNSON, an AT-TLS rule 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. This AT-TLS rule will act as the TLS server in lieu of the application server during the handshake.
When an API requester is uses the options below while running under identity JOHNSON, an AT-TLS rule will be triggered 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")
```

Again, for examples of the steps required to enable security between a z/OS Connect server and subsystems, like Db2 and IMS Connect or from a MVS batch job to a z/OS Connect server, see the security exercises at URL https://tinyurl.com/y3uwq9nl

### 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 are 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.

```plaintext
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.

```plaintext
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.

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

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

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

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

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

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

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

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

```plaintext
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)

  BPXP015I HFS PROGRAM programName IS NOT MARKED PROGRAM CONTROLLED.

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 0x0000004. 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 ATUSERS 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 ATUSERS 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.

```xml
<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 (`clientAuthentication="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**

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).

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 hostname 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 Optimized Local Adapter

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

```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*

```xml
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)
```

WOLA Error Messages

In this section the following configuration for the `zosLocalAdapters` configuration element are in the server.xml file of the Liberty server.

```xml
<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.
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 generated 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 `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 containing an invoke request the service name (see below).

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 **Send** button.

```json
{
    "DPH0XCPI": {
        "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 generated by z/OS Connect tooling. The two products are Postman which is available for downloading from [https://www.getpostman.com/apps](https://www.getpostman.com/apps) and cURL (client URL) which is available for downloading from [https://curl.haxx.se/download.html](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

```
```

{"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}}, "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.
Implementing a z/OS Connect EE Policies

This section provides an example of implementing 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 DFHMIIRS 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.

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

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

```xml
<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.

```xml
<!-- 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:

```
```

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 CORS updates

Occasionally applying service will require changing the CORS configuration in a server’s server.xml file, e.g. the PTF for z/OS Connect V3.00.35. Rather than updating the server.xml of each server, we suggest creating a cors.xml file in a shared location. The contents of the new or updated cors.xml file can be obtained from file server.xml in directory $/usr/lpp/IBM/zosconnect/v3r0/runtime/templates/servers/default after the service has been applied. Then add an <include> statement the server.xml of each server for the common cors.xml file at the same time removing any existing CORS configuration elements. The CORS update will be available the next time the server is restarted. This technique avoids needing to change or modify the CORS elements in every server.xml file for every server.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<server description="CORS include">
    <!-- add cors to allow cross origin access, e.g. when using swagger doc from zOS Connect Enterprise Edition -->
    <cors id="defaultCORSConfig" domain="/"
        allowedOrigins="*"
        allowedMethods="GET, POST, PUT, DELETE, OPTIONS"
        allowedHeaders="Origin, Content-Type, Authorization, Cache-Control, Expires, Pragma"
        allowCredentials="true"
        maxAge="3600"/>
</server>
```

Liberty Environment Variables

- **WLP_LOGGING_CONSOLE_LOGLEVEL** - The logging level used to filter messages written to system streams (STDOUT). The valid values are INFO, AUDIT, WARNING, ERROR, and OFF. By default, the WLP_LOGGING_CONSOLE_LOGLEVEL environment variable is set to AUDIT. Valid options are:
  - **AUDIT** - Audit and warning messages will be written to the system output stream (STDOUT). Error messages will be written to the system output stream (STDERR).
  - **ERROR** - Error messages will be written to the system error stream (STDERR).
  - **INFO** - Info, audit, and warning messages will be written to the system output stream. Error messages will be written to the system error stream (STDERR)
  - **OFF** - No server output is written to system streams (STDOUT). Only JVM output is written to system streams(STDOUT).
• **WARNING** - Warning messages will be written to the system output stream (STDOUT). Error messages will be written to the system error stream (STDERR).

STDOUT and STDERR refer to the DD statements in the server JCL, e.g. spool output.

• **WLP_LOGGING_CONSOLE_FORMAT** - The required format for the console. Valid values are DEV, SIMPLE, or JSON format. By default, WLP_LOGGING_CONSOLE_FORMAT is set to DEV. Valid options are:
  - **DEV** - Use the dev logging format.
  - **JSON** - Use the JSON logging format.
  - **SIMPLE** - Use the simple logging format. As of Liberty release 20.0.0.6 (z/OS Connect V3.034), this format writes the messages to STDOUT and STDERR with time stamps included.

• **WLP_OUTPUT_DIR** - This environment variable can be used to specify an alternative location for server generated output such as logs, the workarea directory, and generated files.

• **WLP_USER_DIR** – This environment variables specifies where the runtime environment looks for shared resources and server definitions.

Environment variables can also be used in the server configuration files. For example the following environment variables are automatically set in a Liberty server.

- **server.config.dir** – whose value will automatically be set to the value of variable WLP_USER_DIR concatenated with the name of the server, e.g. /var/ats/zosconnect/serverName

- **server.output.dir** - whose value will automatically be set to the value of variable WLP_OUTPUT_DIR concatenated with the name of the server, e.g. /var/ats/zosconnect/serverName

See URL [https://tinyurl.com/yyv4tjga](https://tinyurl.com/yyv4tjga) for more information.
Managing a z/OS Connect EE server with the Admin Center

WebSphere Liberty Profile provides an Admin Center feature which provides 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://tinyurl.com/y2ec5m4l

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.

```xml
<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.

```xml
<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`.

```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

```xml
<include location="${server.config.dir}/zc3lab/safSecurity.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.ibm.com:9443/adminCenter](https://wg31.washington.ibm.com: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:**
  ![Design View Image]

- **Source View:**
  ![Source View Image]
• 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.

![Design view of include file](image1.png)

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

![Server Config screen](image2.png)

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=*    //CEEOPTS DD *
//SYSPRINT DD SYSOUT=*   //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.

```plaintext
//ASSEM EXEC PGM=ASMA90,PARM='DECK,NOBJECT'
//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=6&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
//SYSSIN DD *
CEEROPT CSECT
CEEROPT AMODE ANY
CEEROPT RMODE ANY
CEEXOPT POSIX=((ON),OVR),
ENVAR=('BAQURI=wg31.washington.ibm.com',
'BAPORT=9120',
'BAUTHNAME=USER1',
'BAPASSWORD=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 DSN=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.

```plaintext
//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  
    CEEUOPT 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.SYSLIB 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 FGM=GETAPI, PARM='111111'
//STEPLIB DD DISP=SHR, DSN=USER1.ZCEE.LOADLIB
// DD DISP=SHR, DSN=ZCEE30.SBAQLIB
//SYSOUT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//
```
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 interval 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 `updateTrigger` 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.

```xml
<!-- 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"/>
```
z/OS Connect and Data Virtualization Manager

This section shows an example of the connectivity requirements for between a DVM server and a z/OS Connect server. The DVM server should be fully initialized before the client z/OS Connect is started. This sequence is required before the DVM server will establish the WOLA communication area between the two servers. If the z/OS Connect server is started first, it will write error messages indicating that the targeted DVM server is not available.

This section shows an example of what was used in the Washington System Center to configure DVM and z/OS Connect.

**DVM configuration**

Updates to the DVM configuration are done in server initialization member in the data set identified by DD name `SYSEXEC`. The member name is determined by concatenation the server’s subsystem name with the string IN00. So, if the subsystem name is AVZS the server initialization member is the PDS member AZVSIN00 in the data set for DD name `SYSEXEC`.

If the DVM start up procedure has JCL like this example:

```plaintext
//AVZS       PROC SSID=AVZS,  
  //        OPT=INIT,  
  //        TRACE=B,  
  //        MSGPFX=AVZ,  
  //        REG=8M,  
  //        MEM=32G,  
  //        HLQ='DVS'  
  .   .   .  
//SYSEXEC    DD   DISP=SHR,DSN=&HLQ..&SSID..SAVZEXEC
```

The server initialization member would be member `AVZSIN00` in data set `DVS.AZVS.SAVZEXEC`.  

Enable the z/OS Connect interface facility by location the changing the `DontDoThis` check to `DoThis` and providing values for the NAME, RNAME and WNAME ZCPATH attributes.

```c
if DoThis then
  do
    "MODIFY PARM NAME(ZCONNECT) VALUE(YES)"
    "MODIFY PARM NAME(NETWORKBUFFERSIZE) VALUE(96K)"

    "DEFINE ZCPATH",
      "  NAME(ZCEE)",
      "  RNAME(ZCEEDVM)",
      "  WNAME(ZCEEDVM)"
    ""
  end
```
z/OS Connect server.xml configuration

An OMVS file was created named DVM.xml and included in the base server.xml file. The contents of this file are shown below:

```xml
<!-- Enable DVM related features -->
<featureManager>
    <feature>usr:dvsProvider</feature>
    <feature>zosLocalAdapters-1.0</feature>
</featureManager>

<!-- Adapter Details with WOLA Group Name (ZCEEDVM) -->
<zosLocalAdapters wolaName3="NAME3"
wolaName2="NAME2"
wolaGroup="ZCEEDVM"/>

<!-- DVS Service Details with Register Name (ZCEEDVM) -->
<zosconnect_zosConnectService invokeURI="/dvs"
    serviceDescription=""
    serviceRef="dvsService"
    serviceName="dvsService"
    id="zosConnectDvsService"/>

<usr_dvsService invokeURI="/dvs"
    serviceName="DVSS1"
    registerName="ZCEEDVM"
    connectionFactoryRef="wolaCF"
    id="dvsService"/>

<connectionFactory jndiName="eis/ola" id="wolaCF"/>
</connectionFactory>

<zosconnect_zosConnectService serviceRef="svc1"
    serviceAsyncRequestTimeout="600s"
    serviceName="dvs1" id="sdef1"/>

<zosconnect_localAdaptersConnectService
    connectionWaitTimeout="7200"
    connectionFactoryRef="wolaCF"
    serviceName="DVSS1"
    registerName="ZCEEDVM"
    id="svc1"/>
```

Finally, to allow the two servers to connect using WOLA a CBIND RACF resource was defined and the identity under which the z/OS Connect server was running was permitted READ access.

```
RDEFINE CBIND BBG.WOLA.ZCEEDVM.** UACC(NONE)
PERMIT BBG.WOLA.ZCEEDVM.** CLASS(CBIND) ID(LIBSERV) ACC(READ)
SETROPTS RACLIST(CBIND) REFRESH
```
Sample JCL

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

Display Java version

The JCL below can be used to determine the current Java version level and verify the user can instantiate a Java JVM.

```plaintext
//CHKJAVA JOB 'ZCEE',CLASS=A,REGION=0M,MSGCLASS=H,NOTIFY=&SYSUID
//****************************************************************
//*  Step IEBCOPY - Set common environment variables                *
//****************************************************************
//IEBGENER EXEC PGM=IEBGENER                                       
//SYSPRINT DD SYSOUT=*                                            
//SYSUT1 DD *                                                      
JAVA_HOME=/usr/lpp/java/J8.0_64                                   
//SYSUT2 DD DSN=&STDENV,DISP=(,PASS),                             
//   DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),SPACE=(TRK,(1,1))           
//SYSIN DD DUMMY                                                   
//****************************************************************
//*  Step JAVA - Invoke the java -version command                  *
//****************************************************************
//JAVA EXEC PGM=IKJEFT01,REGION=0M                                
//SYSERR DD SYSOUT=*                                              
//STDOUT DD SYSOUT=*                                              
//STDENV DD DSN=&STDENV,DISP=(OLD,DELETE)                         
//SYSTSPRT DD SYSOUT=*                                            
//SYSTSIN DD *                                                    
BPXBATCH SH java -version                                         
```
**Execute the z/OS Connect setup script**

The JCL below can be used to execute the z/OS Connect setup script; this script must be run at least once on each LPAR where a z/OS Connect server will be executing.

```plaintext
//***************************************************************
* Step IEBCOPY - Set common environment variables
//***************************************************************
* IEBGENER EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=* 
//SYSUT1 DD *
JAVA_HOME=/usr/lpp/java/J8.0_64 
ZCEEPATH=/usr/lpp/IBM/zosconnect/v3r0
//SYSUT2 DD DSN=&&STDENV,DISP=(,PASS),
   DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),SPACE=(TRK,(1,1))
//SYSIN DD DUMMY
//***************************************************************
* Step ZCSETUP - Invoke the zconsetup script
//***************************************************************
* ZCSETUP EXEC PGM=IKJEFT01,REGION=0M
//SYSSERR DD SYSOUT=* 
//STDOUT DD SYSOUT=* 
//STDENV DD DSN=&&STDENV,DISP=(OLD,DELETE)
//SYSTSPRT DD SYSOUT=* 
//SYSTSIN DD *
BPXBATCH SH $ZCEEPATH/bin/zconsetup install
```
Creating a z/OS Connect server

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

```plaintext
//ZCEESRVR JOB (0), 'ZCEE DEPLOY', CLASS=A, REGION=0M, MSGCLASS=H, NOTIFY=&SYSUID
//******************************************************************************
//* Step IEBCOPY - Set common environment variables
//******************************************************************************
//IEBGENER EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*  
//SYSUT1 DD *
JAVA_HOME=/usr/lpp/java/J8.0_64
ZCEEPATH=/usr/lpp/IBM/zosconnect/v3r0
SERVER=myServer
TEMPLATE=zosconnect:default
WLP_USER_DIR=/var/zosconnect
USER=LIBSERV
GROUP=LIBGRP
//SYSUT2 DD DSN=&&STDENV,DISP=(,PASS),
   DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),SPACE=(TRK,(1,1))
//SYSSIN DD DUMMY
//******************************************************************************
//* Step ZCEESRVR - Use the zosconnect command to create a server
//******************************************************************************
//ZCEESRVR EXEC PGM=IKJEFT01, REGION=0M
//SYSSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
//STDENV DD DSN=&&STDENV,DISP=(OLD,PASS)
//SYSTSIN DD *
BPXBATCH SH $ZCEEPATH/bin/zosconnect create $SERVER + --template=$TEMPLATE
//******************************************************************************
//* Step CHOWN - Change directory and file ownership
//******************************************************************************
//CHOWN EXEC PGM=IKJEFT01, REGION=0M
//SYSSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
//STDENV DD DSN=&&STDENV,DISP=(OLD,DELETE)
//SYSTSIN DD *
BPXBATCH SH chown -R $USER:$GROUP $WLP_USER_DIR/servers/$SERVER
```
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.

```plaintext
//********************************************************************
//* Step ALLOC    - Use the TSO ALLOCATE command to allocate a PDSE load library */
//********************************************************************
//ALLOC EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*  
//SYSIN DD *  
  DELETE USER1.WOLA2006.LOADLIB  
  SET MAXCC=0  
  ALLOC DSNAME('USER1.WOLA2006.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.WOLA2006.LOADLIB  
//SYSTSPRT DD SYSOUT=*  
//SYSTSIN DD *  
BPXBATCH SH cp -Xv $WLPPATH/wlp/clients/zos/* ""$DSNAME""  
```
Base64 Encoding and Swagger UI

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* and would be entered in the *Authorization* area as *BASIC RnJlZDpmcmVkcHf*. There are several ways to perform this encoding. The URL [https://www.base64encode.org/](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.

```java
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("encoded value is "+new String(bytesEncoded));

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

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).
End of WP102724