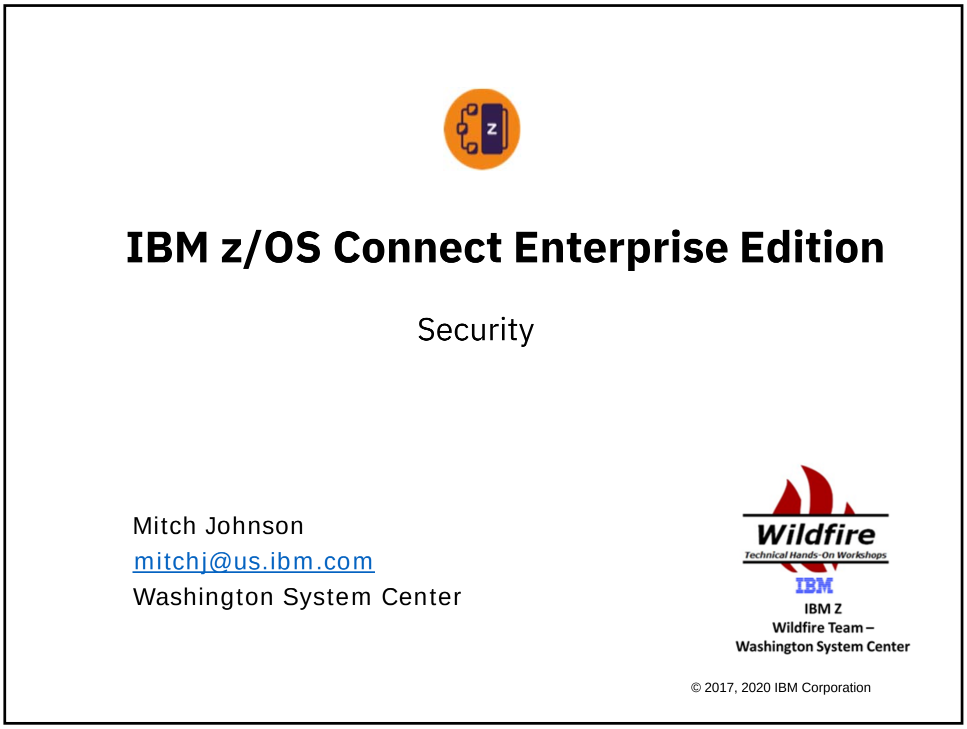


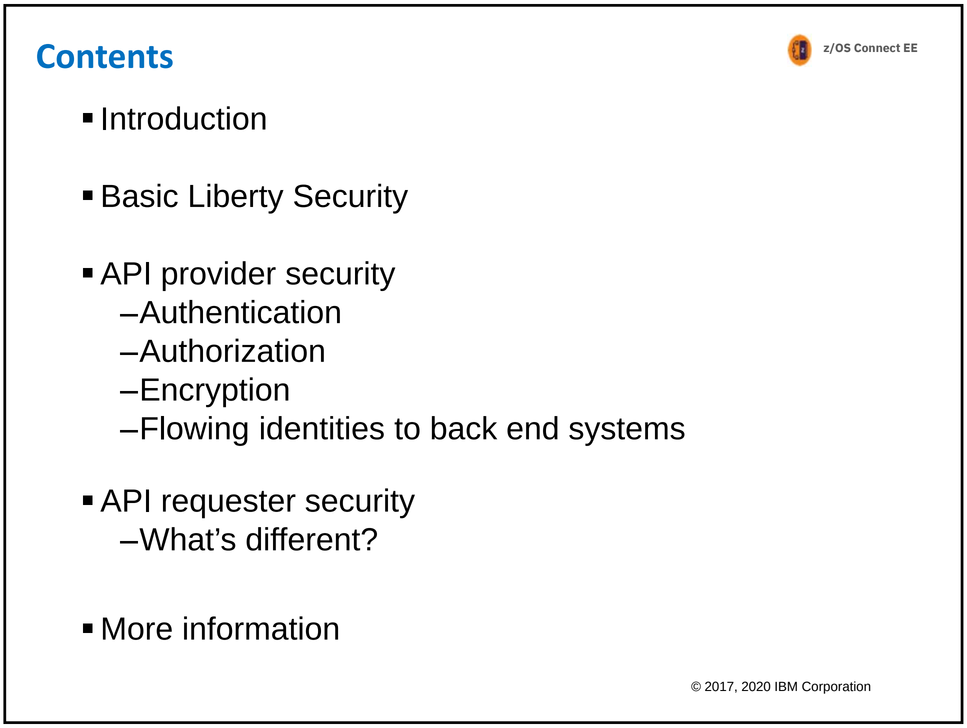
IBM z/OS Connect Enterprise Edition

Security

Mitch Johnson
mitchj@us.ibm.com
Washington System Center

Wildfire
Technical Hands-On Workshops
IBM
IBM Z
Wildfire Team –
Washington System Center

1



Contents

z/OS Connect EE

- Introduction
- Basic Liberty Security
- API provider security
  - –Authentication
  - –Authorization
  - –Encryption
  - –Flowing identities to back end systems
- API requester security
  - –What’s different?
- More information

2

General considerations for securing REST APIs

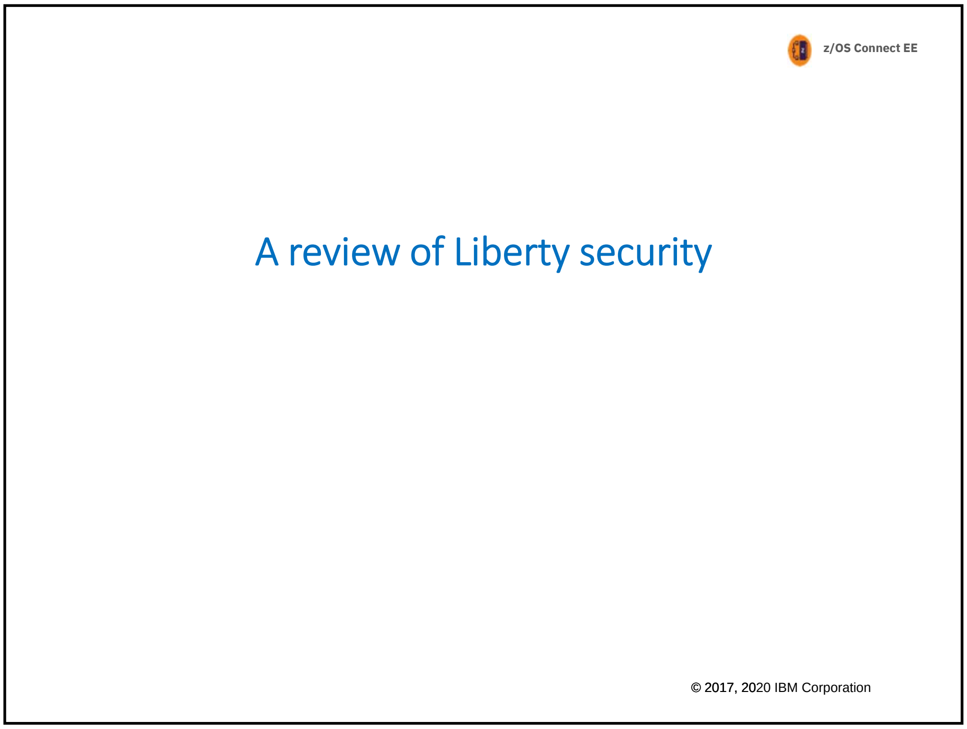
z/OS Connect EE

- Know who is invoking the API (**Authentication)**
- Ensure that the data has not been altered in transit (**Data Integrity)** and ensure confidentiality of data in transit (**Encryption)**
- Control access to APIs (**Authorization)**
  - End user
  - Application
- Know who invoked the APIs (**Audit**)

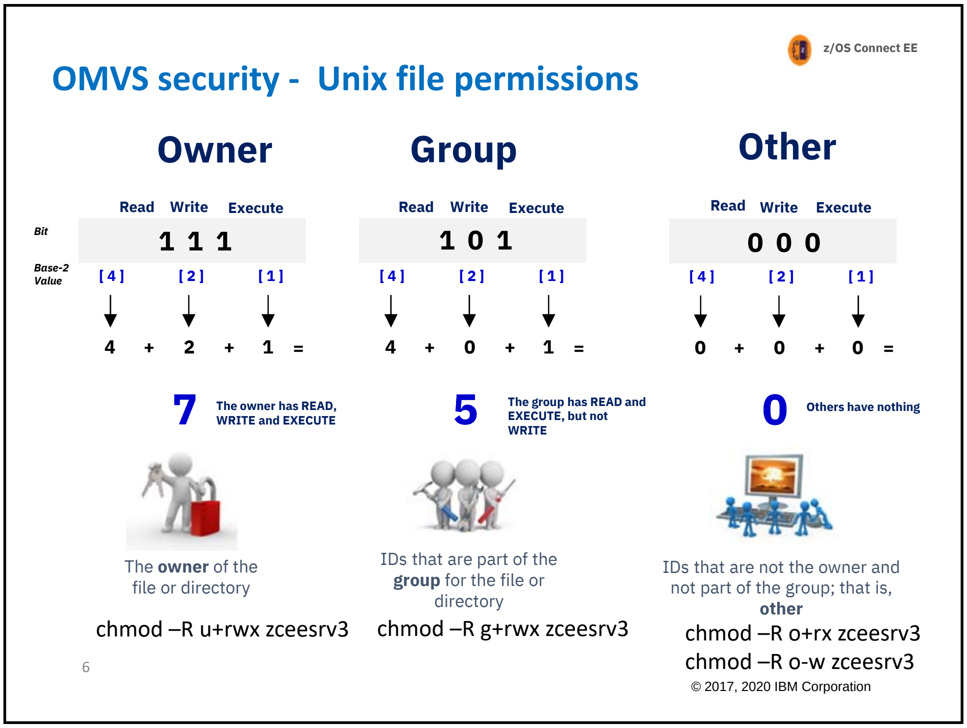
Common challenges

z/OS Connect EE

- **End-to-end security** is hampered by the issue of how to provide secure access between middleware components that use disparate security technologies e.g. registries
  - › This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JWT
- Security when using z/OS Connect is implemented in many products including z/OS Connect, WebSphere Liberty Profile on z/OS, SAF/RACF, CICS, IMS, Db2, MQ …
  - › And these are all documented in different places
- Often security is at odds with **performance**, because the most secure techniques often involve the most processing overhead especially if not configured optimally



A review of Liberty security



OMVS security - Unix file permissions

	Owner			Group			Other		
	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
Bit	1	1	1	1	0	1	0	0	0
Base-2 Value	[4]	[2]	[1]	[4]	[2]	[1]	[4]	[2]	[1]

4 + 2 + 1 = **7** The owner has READ, WRITE and EXECUTE

4 + 0 + 1 = **5** The group has READ and EXECUTE, but not WRITE

0 + 0 + 0 = **0** Others have nothing

The **owner** of the file or directory

chmod –R u+rwx zceesrv3

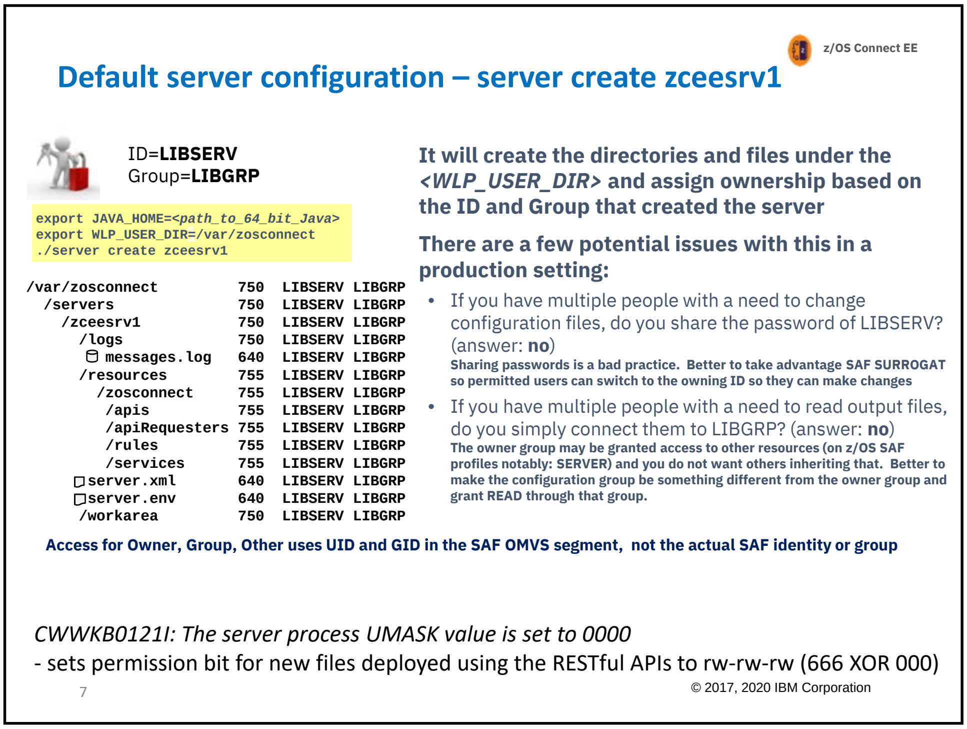
IDs that are part of the **group** for the file or directory

chmod –R g+rwx zceesrv3

IDs that are not the owner and not part of the group; that is, **other**

chmod –R o+rx zceesrv3

chmod –R o-w zceesrv3



z/OS Connect EE

Default server configuration – server create zceesrv1

ID=**LIBSERV**
Group=**LIBGRP**

```
export JAVA_HOME=<path_to_64_bit_Java>
export WLP_USER_DIR=/var/zosconnect
./server create zceesrv1
```

```
/var/zosconnect          750  LIBSERV LIBGRP
  /servers               750  LIBSERV LIBGRP
    /zceesrv1            750  LIBSERV LIBGRP
      /logs              750  LIBSERV LIBGRP
        messages.log     640  LIBSERV LIBGRP
      /resources         755  LIBSERV LIBGRP
        /zosconnect      755  LIBSERV LIBGRP
          /apis          755  LIBSERV LIBGRP
          /apiRequesters 755  LIBSERV LIBGRP
          /rules         755  LIBSERV LIBGRP
          /services      755  LIBSERV LIBGRP
      server.xml         640  LIBSERV LIBGRP
      server.env         640  LIBSERV LIBGRP
      /workarea          750  LIBSERV LIBGRP
```

It will create the directories and files under the *<WLP_USER_DIR>* and assign ownership based on the ID and Group that created the server

There are a few potential issues with this in a production setting:

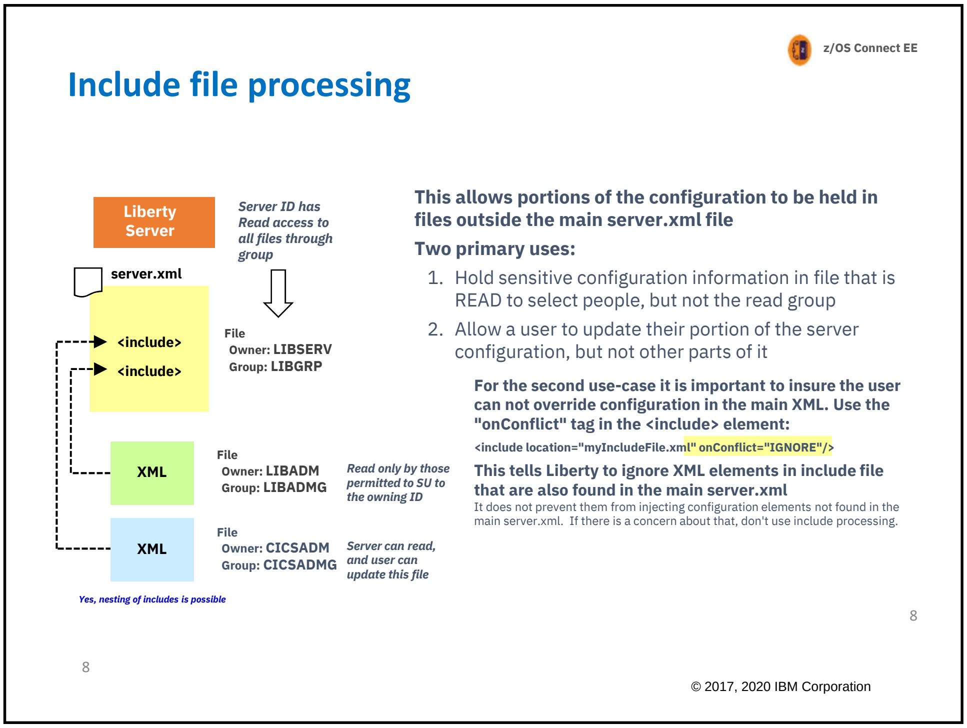
- If you have multiple people with a need to change configuration files, do you share the password of LIBSERV? (answer: **no**)
  Sharing passwords is a bad practice. Better to take advantage SAF SURROGAT so permitted users can switch to the owning ID so they can make changes
- If you have multiple people with a need to read output files, do you simply connect them to LIBGRP? (answer: **no**)
  The owner group may be granted access to other resources (on z/OS SAF profiles notably: SERVER) and you do not want others inheriting that. Better to make the configuration group be something different from the owner group and grant READ through that group.

Access for Owner, Group, Other uses UID and GID in the SAF OMVS segment, not the actual SAF identity or group

CWWKB0121I: The server process UMASK value is set to 0000
- sets permission bit for new files deployed using the RESTful APIs to rw-rw-rw (666 XOR 000)

7

© 2017, 2020 IBM Corporation



z/OS Connect EE

Include file processing

Liberty Server
Server ID has Read access to all files through group
server.xml
<include>
<include>
File
Owner: LIBSERV
Group: LIBGRP
XML
File
Owner: LIBADM
Group: LIBADMG
Read only by those permitted to SU to the owning ID
XML
File
Owner: CICSADM
Group: CICSADMG
Server can read, and user can update this file
Yes, nesting of includes is possible

This allows portions of the configuration to be held in files outside the main server.xml file

Two primary uses:

1. Hold sensitive configuration information in file that is READ to select people, but not the read group
2. Allow a user to update their portion of the server configuration, but not other parts of it

For the second use-case it is important to insure the user can not override configuration in the main XML. Use the "onConflict" tag in the <include> element:

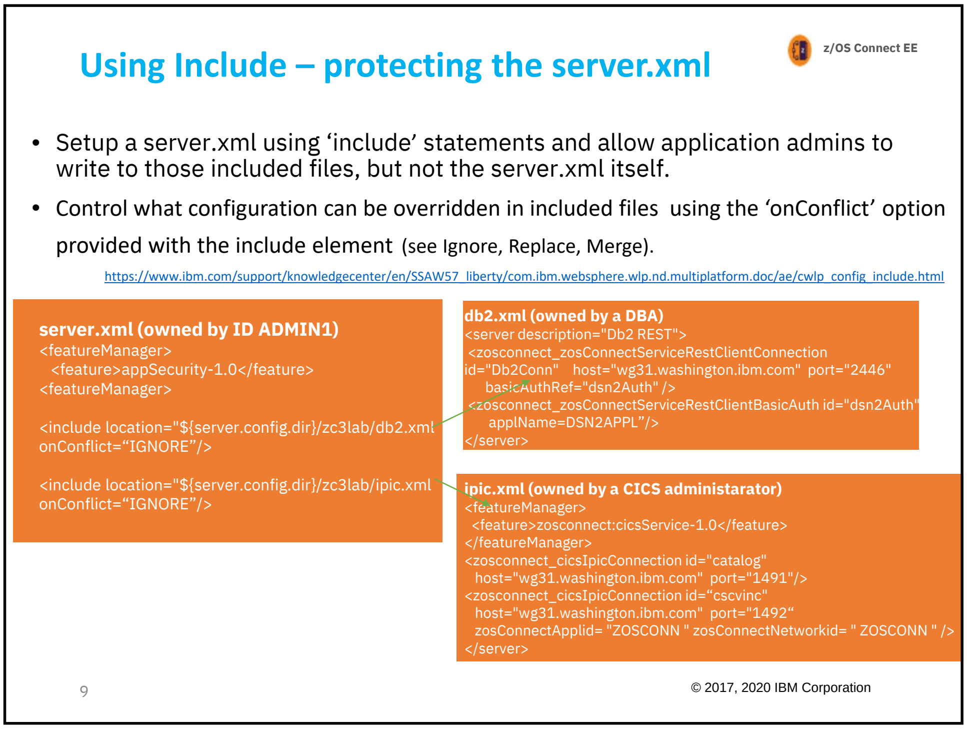
<include location="myIncludeFile.xml" onConflict="IGNORE"/>

This tells Liberty to ignore XML elements in include file that are also found in the main server.xml
It does not prevent them from injecting configuration elements not found in the main server.xml. If there is a concern about that, don't use include processing.

8

8

© 2017, 2020 IBM Corporation



Using Include – protecting the server.xml

z/OS Connect EE

- Setup a server.xml using 'include' statements and allow application admins to write to those included files, but not the server.xml itself.
- Control what configuration can be overridden in included files using the 'onConflict' option provided with the include element (see Ignore, Replace, Merge).

https://www.ibm.com/support/knowledgecenter/en/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_config_include.html

server.xml (owned by ID ADMIN1)

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

<include location="${server.config.dir}/zc3lab/db2.xml
onConflict="IGNORE"/>

<include location="${server.config.dir}/zc3lab/ipic.xml
onConflict="IGNORE"/>
```

db2.xml (owned by a DBA)

```
<server description="Db2 REST">
 <zosconnect_zosConnectServiceRestClientConnection
id="Db2Conn"   host="wg31.washington.ibm.com"  port="2446"
    basicAuthRef="dsn2Auth" />
 <zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
     applName=DSN2APPL"/>
</server>
```

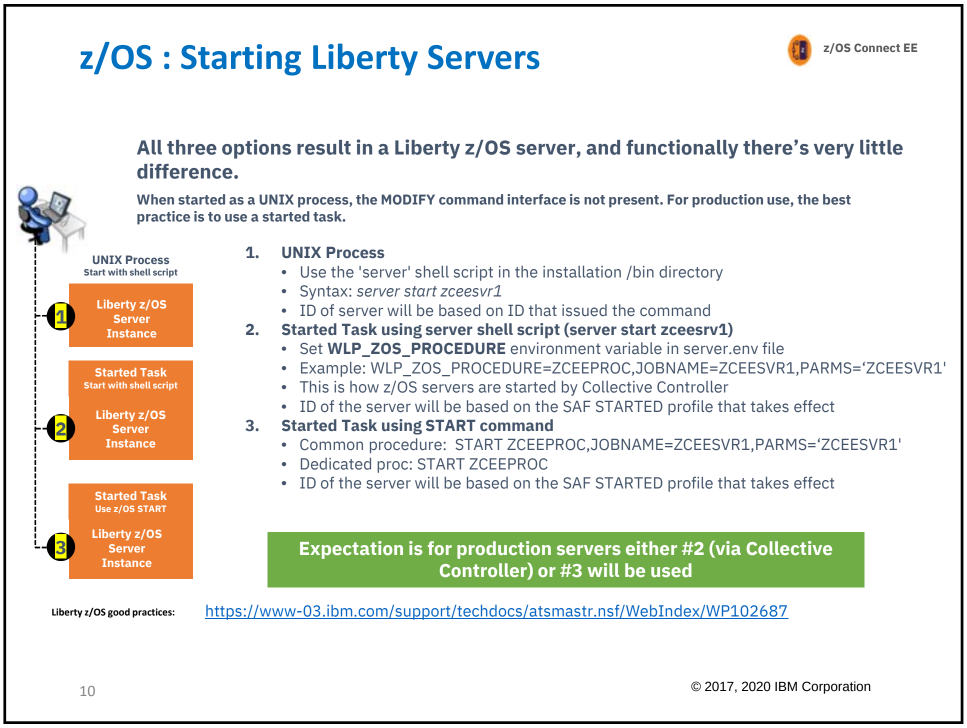
ipic.xml (owned by a CICS administarator)

```
<featureManager>
  <feature>zosconnect:cicsService-1.0</feature>
</featureManager>
<zosconnect_cicsIpicConnection id="catalog"
  host="wg31.washington.ibm.com"  port="1491"/>
<zosconnect_cicsIpicConnection id="cscvinc"
  host="wg31.washington.ibm.com"  port="1492"
  zosConnectApplid= "ZOSCONN " zosConnectNetworkid= " ZOSCONN " />
</server>
```

9

© 2017, 2020 IBM Corporation

9



z/OS : Starting Liberty Servers

z/OS Connect EE

All three options result in a Liberty z/OS server, and functionally there's very little difference.

When started as a UNIX process, the MODIFY command interface is not present. For production use, the best practice is to use a started task.

UNIX Process
Start with shell script
1
Liberty z/OS Server Instance
Started Task
Start with shell script
2
Liberty z/OS Server Instance
Started Task
Use z/OS START
3
Liberty z/OS Server Instance

1. **UNIX Process**
   - Use the 'server' shell script in the installation /bin directory
   - Syntax: *server start zceesvr1*
   - ID of server will be based on ID that issued the command
2. **Started Task using server shell script (server start zceesrv1)**
   - Set **WLP_ZOS_PROCEDURE** environment variable in server.env file
   - Example: WLP_ZOS_PROCEDURE=ZCEEPROC,JOBNAME=ZCEESVR1,PARMS='ZCEESVR1'
   - This is how z/OS servers are started by Collective Controller
   - ID of the server will be based on the SAF STARTED profile that takes effect
3. **Started Task using START command**
   - Common procedure: START ZCEEPROC,JOBNAME=ZCEESVR1,PARMS='ZCEESVR1'
   - Dedicated proc: START ZCEEPROC
   - ID of the server will be based on the SAF STARTED profile that takes effect

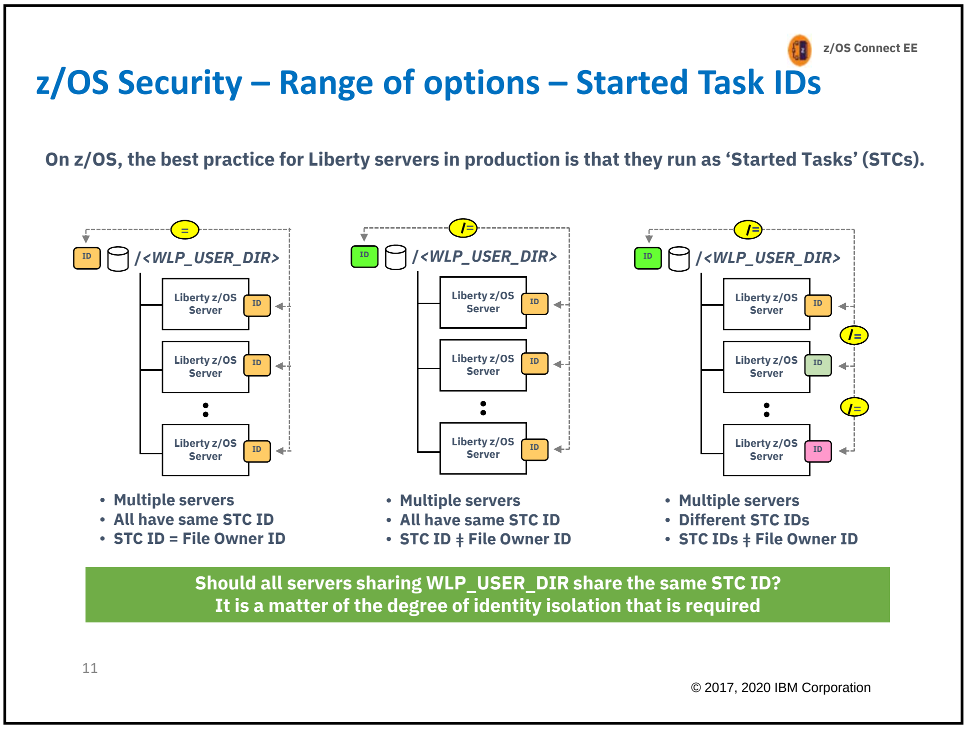
Expectation is for production servers either #2 (via Collective Controller) or #3 will be used

Liberty z/OS good practices: https://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102687

10

© 2017, 2020 IBM Corporation

10



z/OS Connect EE

z/OS Security – Range of options – Started Task IDs

On z/OS, the best practice for Liberty servers in production is that they run as 'Started Tasks' (STCs).

- Multiple servers
- All have same STC ID
- STC ID = File Owner ID

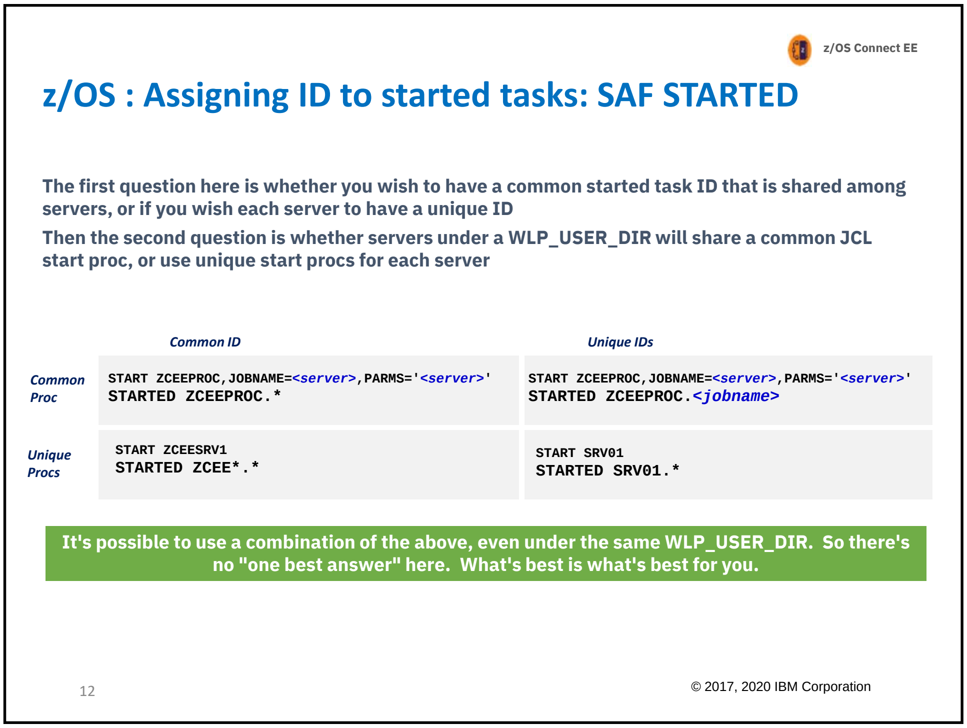
- Multiple servers
- All have same STC ID
- STC ID ǂ File Owner ID

- Multiple servers
- Different STC IDs
- STC IDs ǂ File Owner ID

Should all servers sharing WLP_USER_DIR share the same STC ID?
It is a matter of the degree of identity isolation that is required

11

11



z/OS Connect EE

z/OS : Assigning ID to started tasks: SAF STARTED

The first question here is whether you wish to have a common started task ID that is shared among servers, or if you wish each server to have a unique ID

Then the second question is whether servers under a WLP_USER_DIR will share a common JCL start proc, or use unique start procs for each server

	Common ID	*Unique IDs*

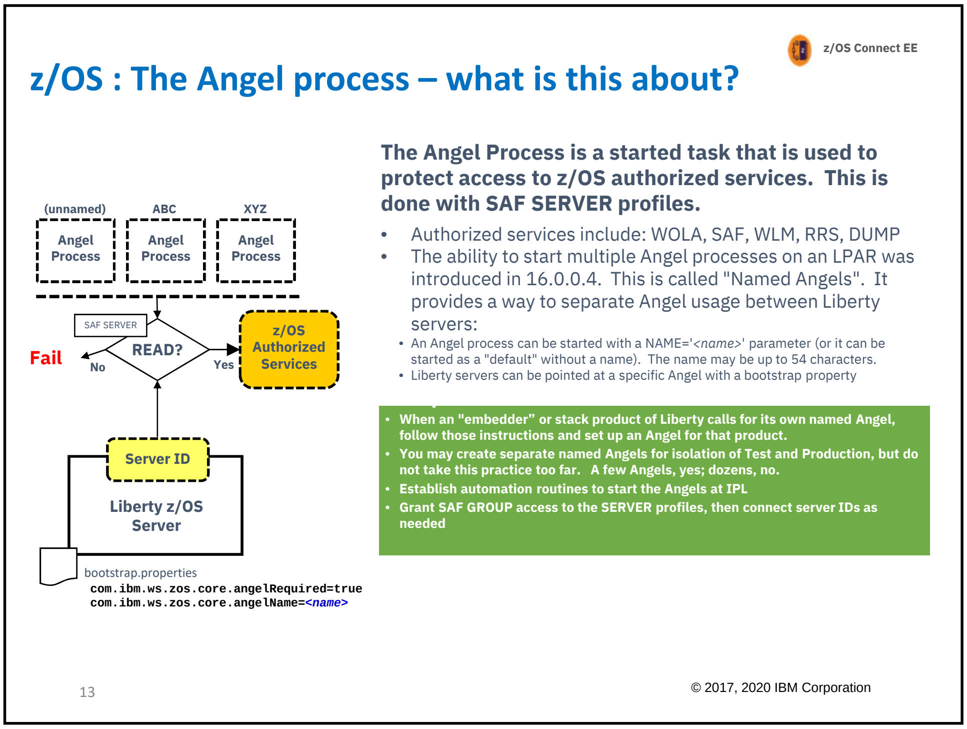
Common Proc	`START ZCEEPROC,JOBNAME=<server>,PARMS='<server>'`
	`STARTED ZCEEPROC.*`
		`START ZCEEPROC,JOBNAME=<server>,PARMS='<server>'`
		`STARTED ZCEEPROC.<jobname>`

Unique Procs	`START ZCEESRV1`
	`STARTED ZCEE*.*`
		`START SRV01`
		`STARTED SRV01.*`

It's possible to use a combination of the above, even under the same WLP_USER_DIR. So there's no "one best answer" here. What's best is what's best for you.

12

12



z/OS Connect EE

z/OS : The Angel process – what is this about?

(unnamed)
ABC
XYZ
Angel Process
Angel Process
Angel Process
SAF SERVER
READ?
Fail
No
Yes
z/OS Authorized Services
Server ID
Liberty z/OS Server
bootstrap.properties

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

The Angel Process is a started task that is used to protect access to z/OS authorized services. This is done with SAF SERVER profiles.

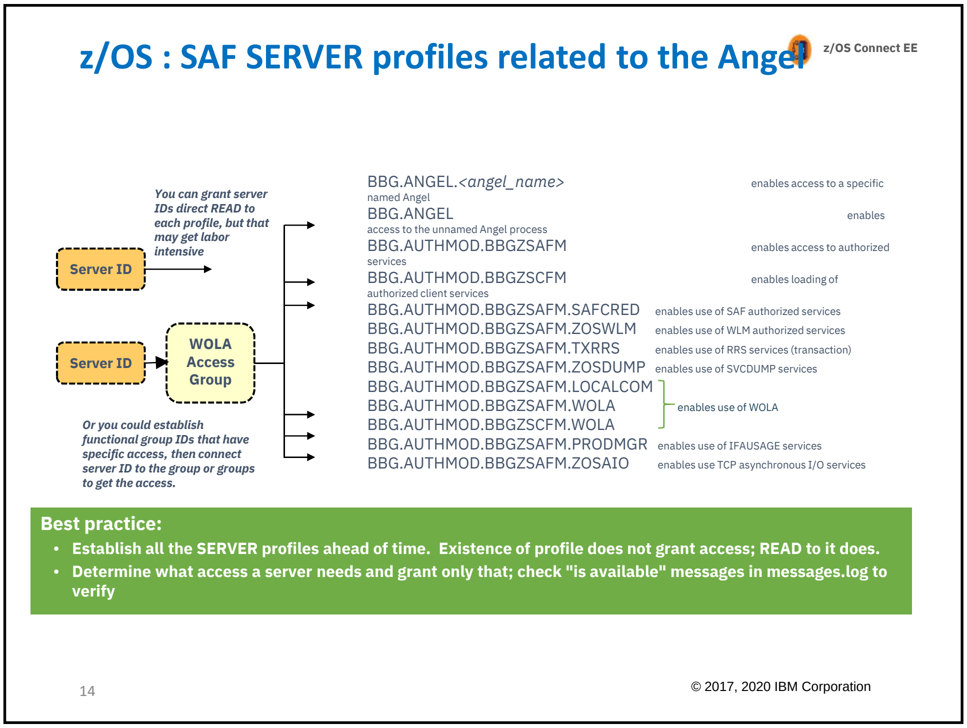
- Authorized services include: WOLA, SAF, WLM, RRS, DUMP
- The ability to start multiple Angel processes on an LPAR was introduced in 16.0.0.4. This is called "Named Angels". It provides a way to separate Angel usage between Liberty servers:
  - An Angel process can be started with a NAME='*<name>*' parameter (or it can be started as a "default" without a name). The name may be up to 54 characters.
  - Liberty servers can be pointed at a specific Angel with a bootstrap property

- **When an "embedder" or stack product of Liberty calls for its own named Angel, follow those instructions and set up an Angel for that product.**
- **You may create separate named Angels for isolation of Test and Production, but do not take this practice too far. A few Angels, yes; dozens, no.**
- **Establish automation routines to start the Angels at IPL**
- **Grant SAF GROUP access to the SERVER profiles, then connect server IDs as needed**

13

© 2017, 2020 IBM Corporation

13



z/OS Connect EE

z/OS : SAF SERVER profiles related to the Angel

Server ID
Server ID
WOLA Access Group

You can grant server IDs direct READ to each profile, but that may get labor intensive

Or you could establish functional group IDs that have specific access, then connect server ID to the group or groups to get the access.

BBG.ANGEL.*<angel_name>*	enables access to a specific named Angel
BBG.ANGEL	enables access to the unnamed Angel process
BBG.AUTHMOD.BBGZSAFM	enables access to authorized services
BBG.AUTHMOD.BBGZSCFM	enables loading of authorized client services
BBG.AUTHMOD.BBGZSAFM.SAFCRED	enables use of SAF authorized services
BBG.AUTHMOD.BBGZSAFM.ZOSWLM	enables use of WLM authorized services
BBG.AUTHMOD.BBGZSAFM.TXRRS	enables use of RRS services (transaction)
BBG.AUTHMOD.BBGZSAFM.ZOSDUMP	enables use of SVCDUMP services
BBG.AUTHMOD.BBGZSAFM.LOCALCOM	enables use of WOLA
BBG.AUTHMOD.BBGZSAFM.WOLA	
BBG.AUTHMOD.BBGZSCFM.WOLA	
BBG.AUTHMOD.BBGZSAFM.PRODMGR	enables use of IFAUSAGE services
BBG.AUTHMOD.BBGZSAFM.ZOSAIO	enables use TCP asynchronous I/O services

Best practice:

- **Establish all the SERVER profiles ahead of time. Existence of profile does not grant access; READ to it does.**
- **Determine what access a server needs and grant only that; check "is available" messages in messages.log to verify**

14

© 2017, 2020 IBM Corporation

14

z/OS Connect EE

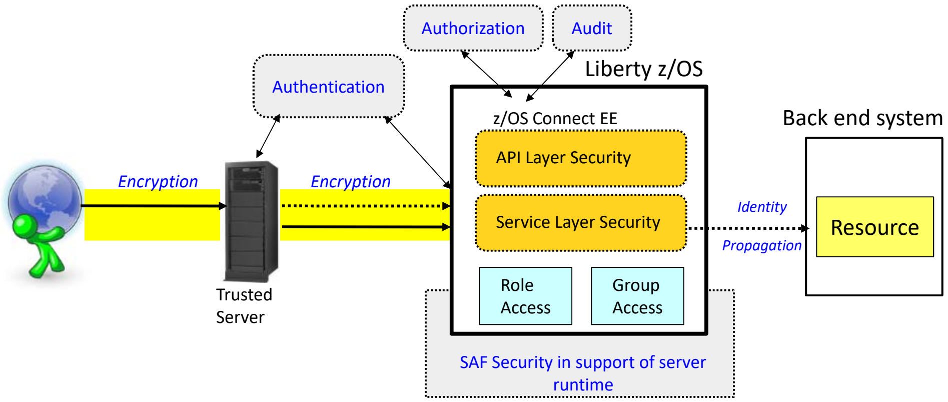
z/OS Connect Security

Overview

15

z/OS Connect EE API provider security overview

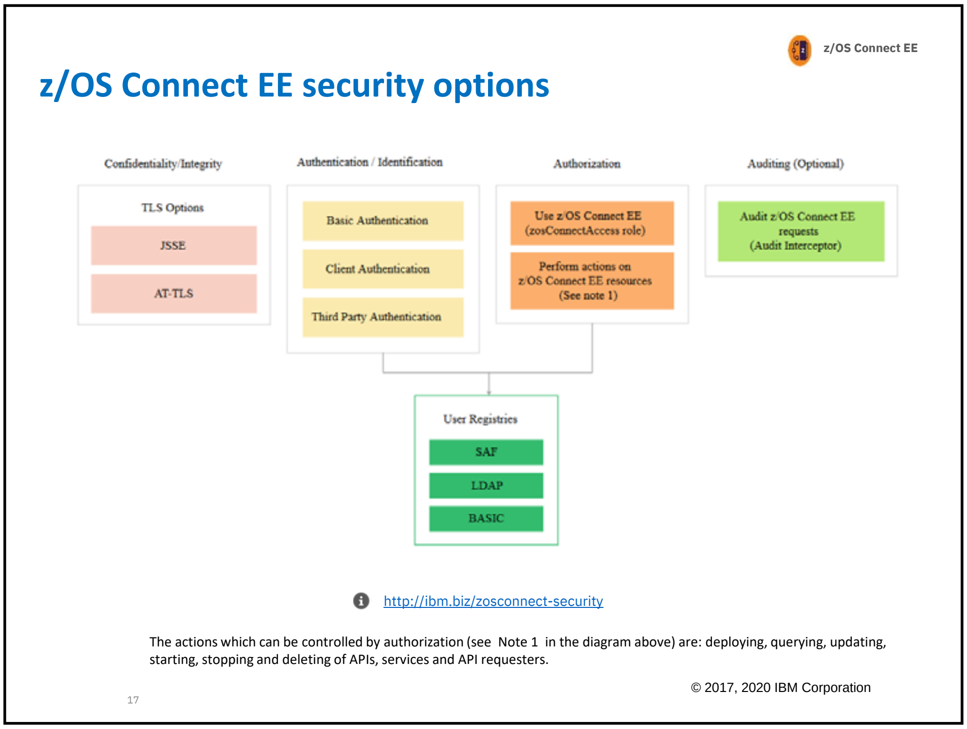
z/OS Connect EE



1. Authentication (basic, client certificates, 3rd party authentication)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (role and group access)
4. Audit
5. Configuring security with SAF
6. Back end identity propagation (CICS, IMS, Db2, MQ)

See Dev Center article "Securing APIs with z/OS Connect EE" overview of z/OS Connect EE security

16



z/OS Connect EE

z/OS Connect EE security options

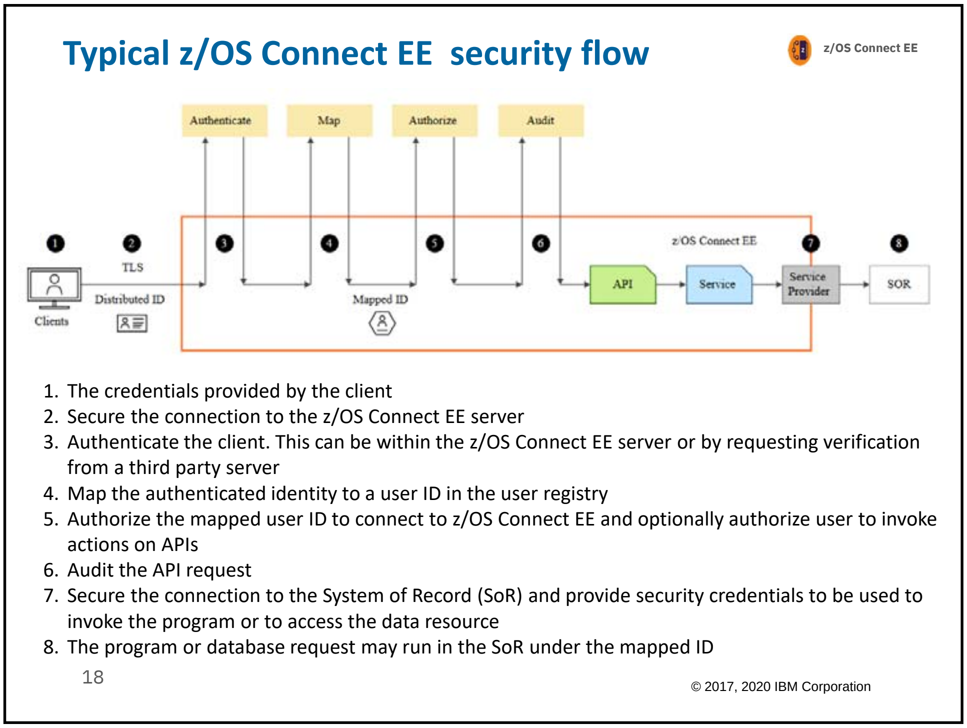
Confidentiality/Integrity
TLS Options
JSSE
AT-TLS
Authentication / Identification
Basic Authentication
Client Authentication
Third Party Authentication
Authorization
Use z/OS Connect EE (zosConnectAccess role)
Perform actions on z/OS Connect EE resources (See note 1)
Auditing (Optional)
Audit z/OS Connect EE requests (Audit Interceptor)
User Registries
SAF
LDAP
BASIC

http://ibm.biz/zosconnect-security

The actions which can be controlled by authorization (see Note 1 in the diagram above) are: deploying, querying, updating, starting, stopping and deleting of APIs, services and API requesters.

17

17



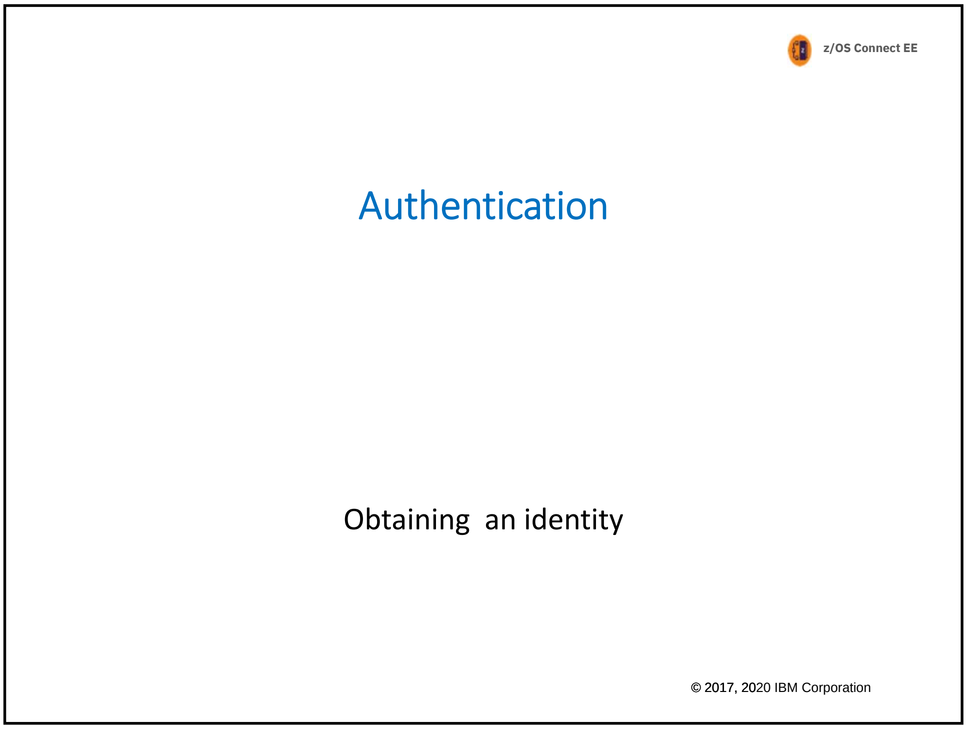
z/OS Connect EE

Typical z/OS Connect EE security flow

1. The credentials provided by the client
2. Secure the connection to the z/OS Connect EE server
3. Authenticate the client. This can be within the z/OS Connect EE server or by requesting verification from a third party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize the mapped user ID to connect to z/OS Connect EE and optionally authorize user to invoke actions on APIs
6. Audit the API request
7. Secure the connection to the System of Record (SoR) and provide security credentials to be used to invoke the program or to access the data resource
8. The program or database request may run in the SoR under the mapped ID

18

18

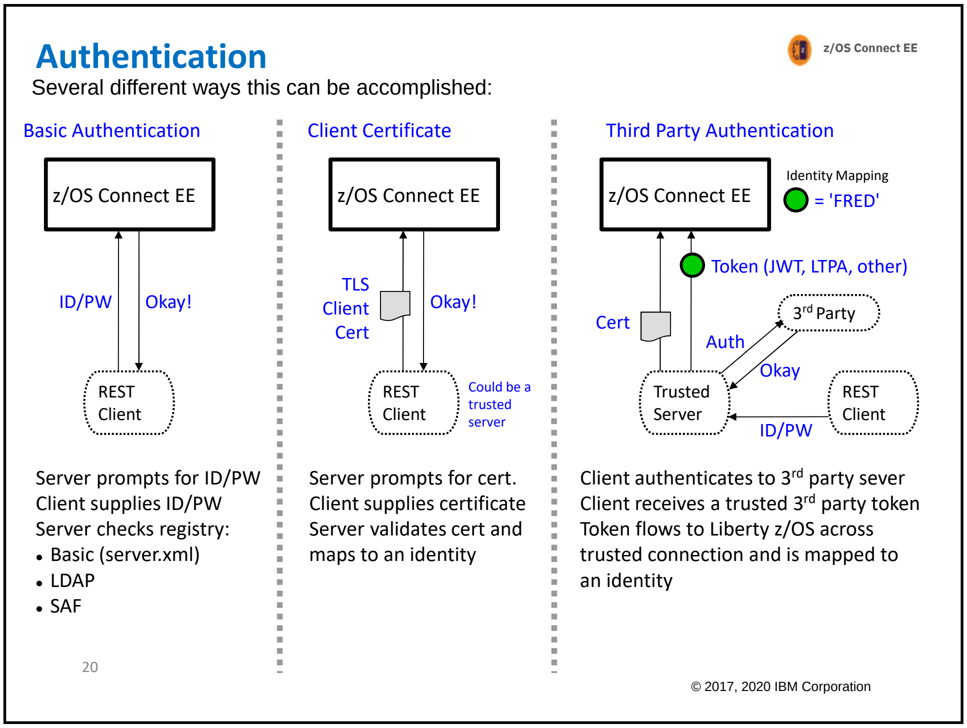


z/OS Connect EE

Authentication

Obtaining an identity

19



z/OS Connect EE

Authentication

Several different ways this can be accomplished:

Basic Authentication

z/OS Connect EE

ID/PW

Okay!

REST Client

Server prompts for ID/PW
Client supplies ID/PW
Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

Client Certificate

z/OS Connect EE

TLS Client Cert

Okay!

REST Client

Could be a trusted server

Server prompts for cert.
Client supplies certificate
Server validates cert and maps to an identity

Third Party Authentication

z/OS Connect EE

Identity Mapping

● = 'FRED'

● Token (JWT, LTPA, other)

Cert

3rd Party

Auth

Okay

Trusted Server

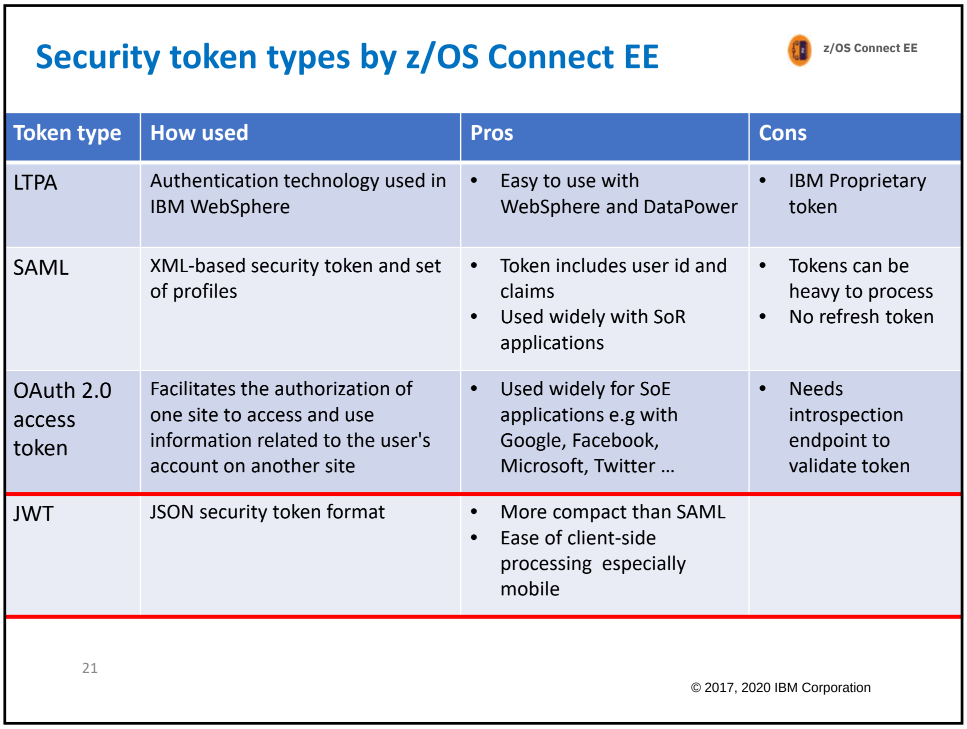
REST Client

ID/PW

Client authenticates to 3rd party sever
Client receives a trusted 3rd party token
Token flows to Liberty z/OS across trusted connection and is mapped to an identity

20

20



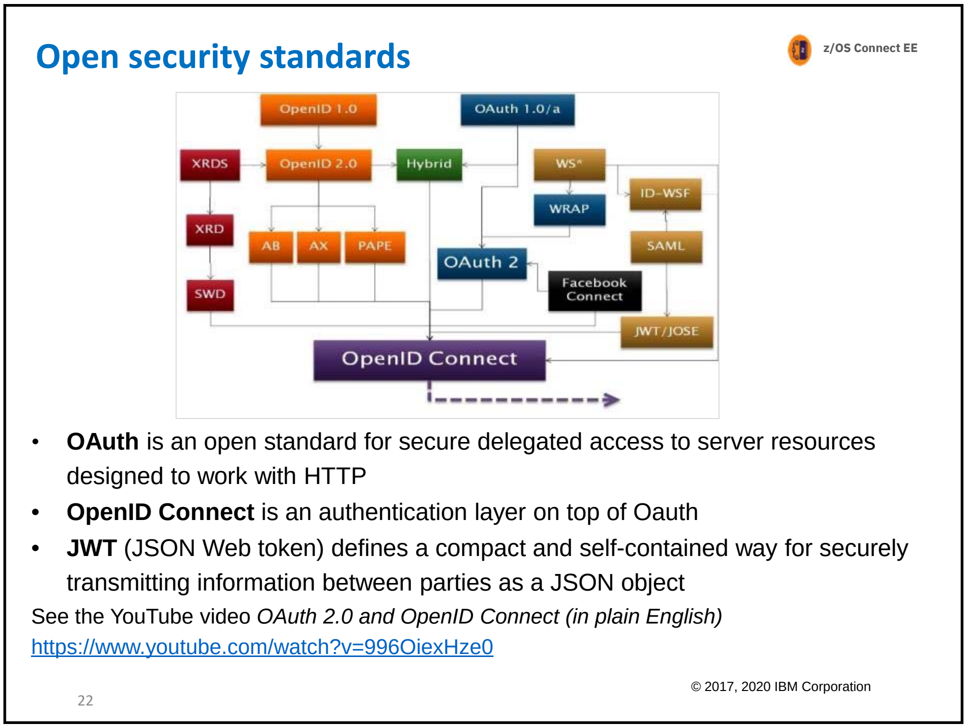
Security token types by z/OS Connect EE

z/OS Connect EE

Token type	How used	Pros	Cons
LTPA	Authentication technology used in IBM WebSphere	• Easy to use with WebSphere and DataPower	• IBM Proprietary token
SAML	XML-based security token and set of profiles	• Token includes user id and claims • Used widely with SoR applications	• Tokens can be heavy to process • No refresh token
OAuth 2.0 access token	Facilitates the authorization of one site to access and use information related to the user's account on another site	• Used widely for SoE applications e.g with Google, Facebook, Microsoft, Twitter …	• Needs introspection endpoint to validate token
JWT	JSON security token format	• More compact than SAML • Ease of client-side processing especially mobile	

21

21



Open security standards

z/OS Connect EE

- **OAuth** is an open standard for secure delegated access to server resources designed to work with HTTP
- **OpenID Connect** is an authentication layer on top of Oauth
- **JWT** (JSON Web token) defines a compact and self-contained way for securely transmitting information between parties as a JSON object

See the YouTube video *OAuth 2.0 and OpenID Connect (in plain English)* https://www.youtube.com/watch?v=996OiexHze0

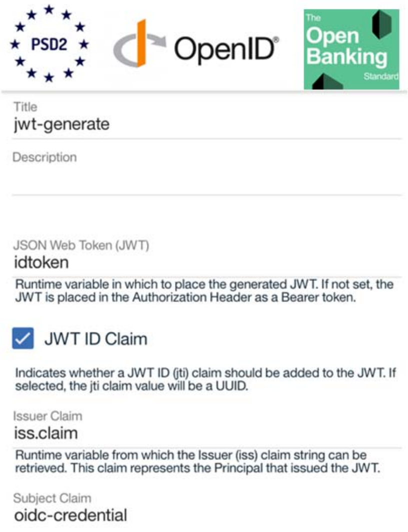
22

22

OpenID Connect Overview

z/OS Connect EE

- **OpenID Connect** (OIDC) is built on top of OAuth 2.0
- Flexible user authentication for Single Sign-On (SSO) to Web, mobile and API workloads
- Addresses European **PSD2** and UK **OpenBanking** requirements for authorization and authentication



Title
jwt-generate

Description

JSON Web Token (JWT)
idtoken
Runtime variable in which to place the generated JWT. If not set, the JWT is placed in the Authorization Header as a Bearer token.

☑ JWT ID Claim

Indicates whether a JWT ID (jti) claim should be added to the JWT. If selected, the jti claim value will be a UUID.

Issuer Claim
iss.claim
Runtime variable from which the Issuer (iss) claim string can be retrieved. This claim represents the Principal that issued the JWT.

Subject Claim
oidc-credential

Why JWT with z/OS Connect EE?

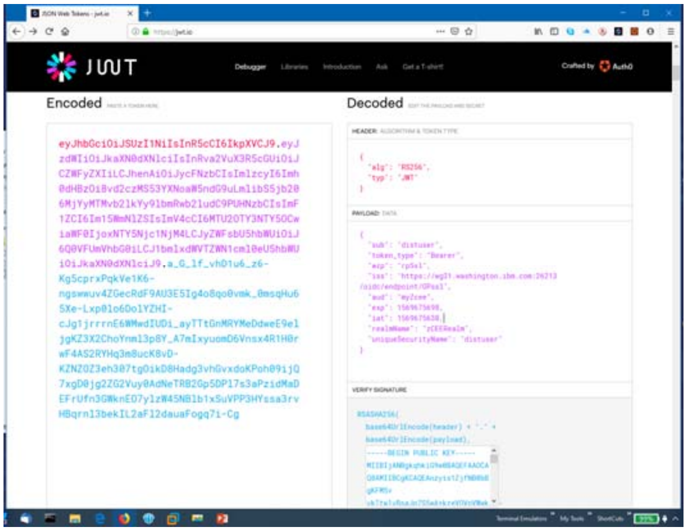
z/OS Connect EE

- Token validation does **not** require an additional trip and can be validated locally by z/OS Connect server
- Parties can easily agree on a specific set of **custom** claims in order to exchange both authentication and authorization information
- Widely adopted by different Single Sign-On solutions and well known standards such as **OpenID Connect**
- **Message-level** security using signature standard
- JWT tokens are **lighter** weight than other XML based tokens e.g SAML

JWT (JSON Web Token)

z/OS Connect EE

- JWT is a compact way of representing claims that are to be transferred between two parties
- Normally transmitted via HTTP header
- Consists of three parts
  - Header
  - Payload
  - Signature



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiJkaXN0dXNlciIsInRva2VuX3R5cGUiOiJ
CZWFyZXIiLCJhenAiOiJycFNzbCIsImlzcyI6Imh
0dHBzOi8vd2czMS53YXNoaW5ndG9uLmlibS5jb20
6MjYyMTMvb2lkYy9lbmRwb2ludC9PUHNzbCIsImF
1ZCI6Im15WmNlZSIsImV4cCI6MTU2OTY3NTY5OCw
iaWF0IjoxNTY5Njc1NjM4LCJyZWFsbU5hbWUiOiJ
6Q0VFUmVhbG0iLCJ1bmlxdWVTZWN1cml0eU5hbWU
iOiJkaXN0dXNlciJ9.a_G_lf_vhD1u6_z6-
Kg5cprxPqkVe1K6-
ngswwuv4ZGecRdF9AU3E5Ig4o8qo0vmk_0msqHu6
5Xe-Lxp0lo6Do1YZHI-
cJg1jrrrnE6WMwdIUDi_ayTTtGnMRYMeDdweE9el
jgKZ3X2ChoYnml3p8Y_A7mIxyuomD6Vnsx4R1H0r
wF4AS2RYHq3m8ucK8vD-
KZNZOZ3eh307tgOikD8Hadg3vhGvxdoKPoh09ijQ
7xgD0jg2ZG2Vuy0AdNeTRB2Gp5DP17s3aPzidMaD
EFrUfn3GWknED7ylzW45NB1b1xSuVPP3HYssa3rv
HBqrnl3bekIL2aF12dauaFogq7i-Cg
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

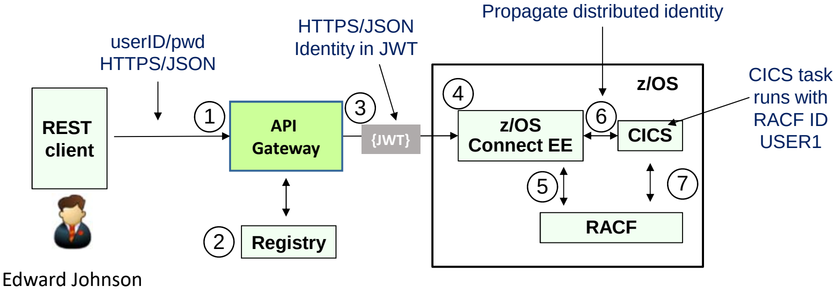
```
{
  "sub": "distuser",
  "token_type": "Bearer",
  "azp": "rpSsl",
  "iss": "https://wg31.washington.ibm.com:26213
/oidc/endpoint/OPssl",
  "aud": "myZcee",
  "exp": 1569675698,
  "iat": 1569675638,
  "realmName": "zCEERealm",
  "uniqueSecurityName": "distuser"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCA
  Q8AMIIBCgKCAQEAnzyis1ZjfNB0bB
  gKFMSv
  vkTtwlvBsaJq7S5wA+kzeVOVpVWwk
```

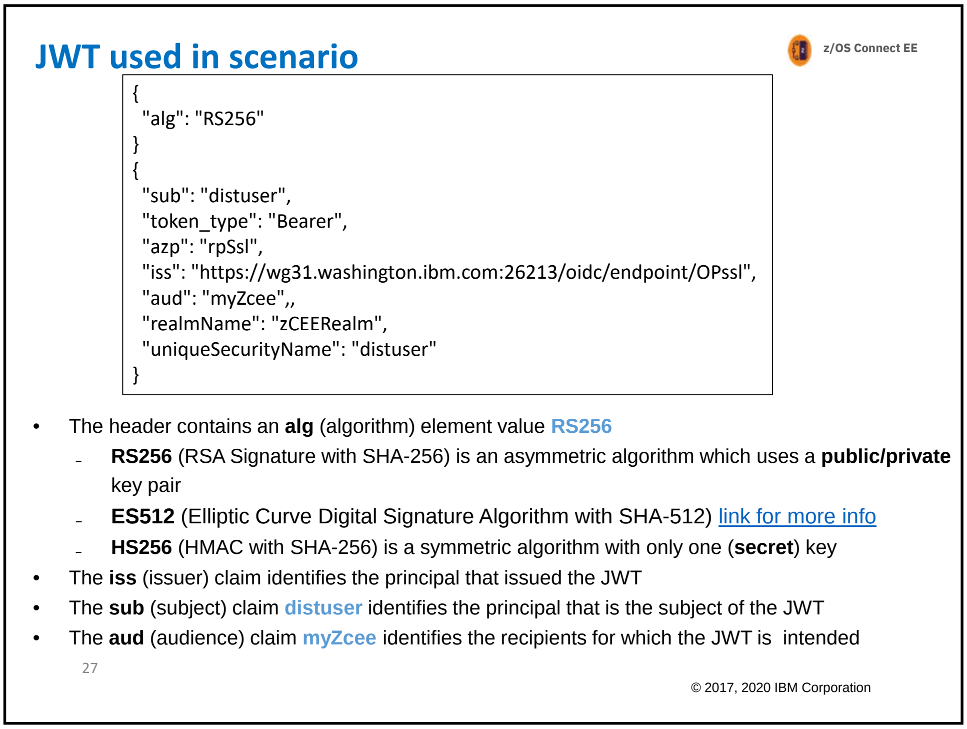
Example scenario – security flow

z/OS Connect EE



RACMAP ID(USER1) MAP USERDIDFILTER(NAME(‘distuser‘)) REGISTRY(NAME('*'))

1. User authenticates with the managed API using a "distributed" identity and a password
2. An external registry is used as the user registry for distributed users and groups
3. API Gateway generates a JWT and forwards the token with the request to z/OS Connect EE
4. z/OS Connect EE validates JWT
5. z/OS Connect EE calls RACF to map distributed ID to RACF user ID and authorizes access to API
6. z/OS Connect EE CICS service provider propagates distributed ID to CICS
7. CICS calls RACF to map distributed ID to RACF user ID and performs resource authorization checks



JWT used in scenario

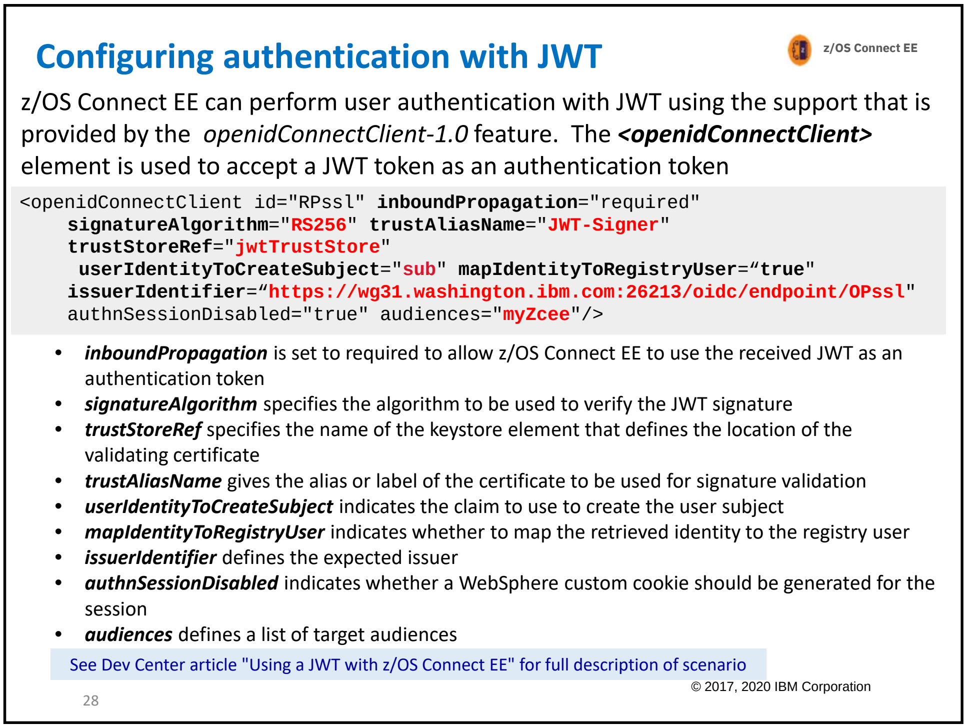
z/OS Connect EE

```
{
  "alg": "RS256"
}
{
  "sub": "distuser",
  "token_type": "Bearer",
  "azp": "rpSsl",
  "iss": "https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl",
  "aud": "myZcee",,
  "realmName": "zCEERealm",
  "uniqueSecurityName": "distuser"
}
```

- The header contains an **alg** (algorithm) element value **RS256**
  - **RS256** (RSA Signature with SHA-256) is an asymmetric algorithm which uses a **public/private** key pair
  - **ES512** (Elliptic Curve Digital Signature Algorithm with SHA-512) link for more info
  - **HS256** (HMAC with SHA-256) is a symmetric algorithm with only one (**secret**) key
- The **iss** (issuer) claim identifies the principal that issued the JWT
- The **sub** (subject) claim **distuser** identifies the principal that is the subject of the JWT
- The **aud** (audience) claim **myZcee** identifies the recipients for which the JWT is intended

27

27



Configuring authentication with JWT

z/OS Connect EE

z/OS Connect EE can perform user authentication with JWT using the support that is provided by the *openidConnectClient-1.0* feature. The ***<openidConnectClient>*** element is used to accept a JWT token as an authentication token

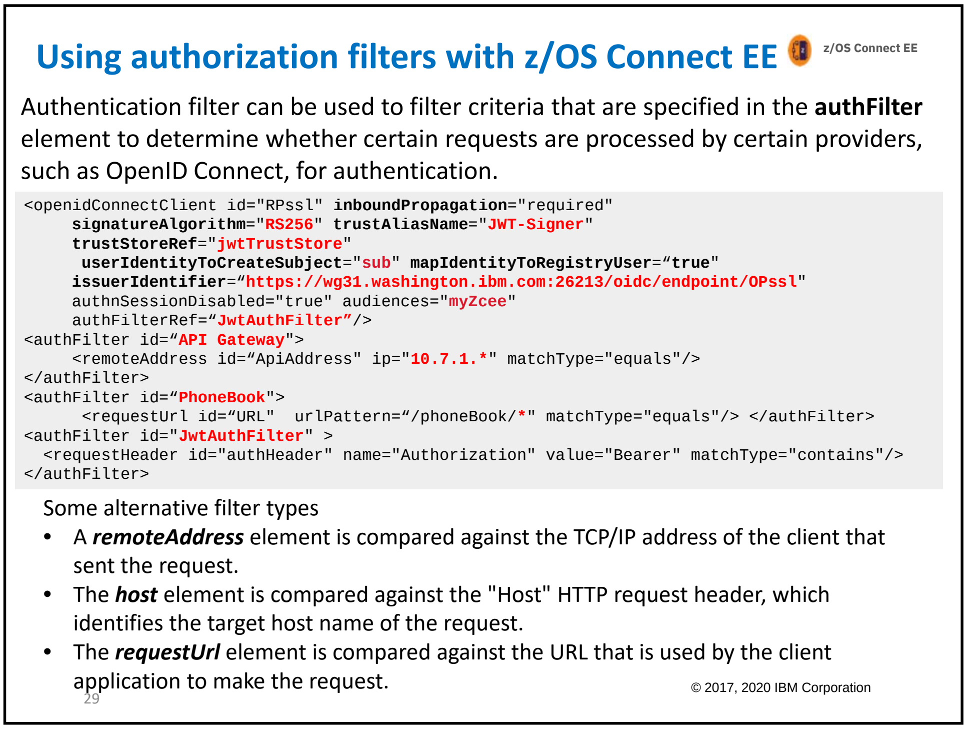
```
<openidConnectClient id="RPssl" inboundPropagation="required"
    signatureAlgorithm="RS256" trustAliasName="JWT-Signer"
    trustStoreRef="jwtTrustStore"
     userIdentityToCreateSubject="sub" mapIdentityToRegistryUser=“true"
    issuerIdentifier=“https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl"
    authnSessionDisabled="true" audiences="myZcee"/>
```

- ***inboundPropagation*** is set to required to allow z/OS Connect EE to use the received JWT as an authentication token
- ***signatureAlgorithm*** specifies the algorithm to be used to verify the JWT signature
- ***trustStoreRef*** specifies the name of the keystore element that defines the location of the validating certificate
- ***trustAliasName*** gives the alias or label of the certificate to be used for signature validation
- ***userIdentityToCreateSubject*** indicates the claim to use to create the user subject
- ***mapIdentityToRegistryUser*** indicates whether to map the retrieved identity to the registry user
- ***issuerIdentifier*** defines the expected issuer
- ***authnSessionDisabled*** indicates whether a WebSphere custom cookie should be generated for the session
- ***audiences*** defines a list of target audiences

See Dev Center article "Using a JWT with z/OS Connect EE" for full description of scenario

28

28



Using authorization filters with z/OS Connect EE

z/OS Connect EE

Authentication filter can be used to filter criteria that are specified in the **authFilter** element to determine whether certain requests are processed by certain providers, such as OpenID Connect, for authentication.

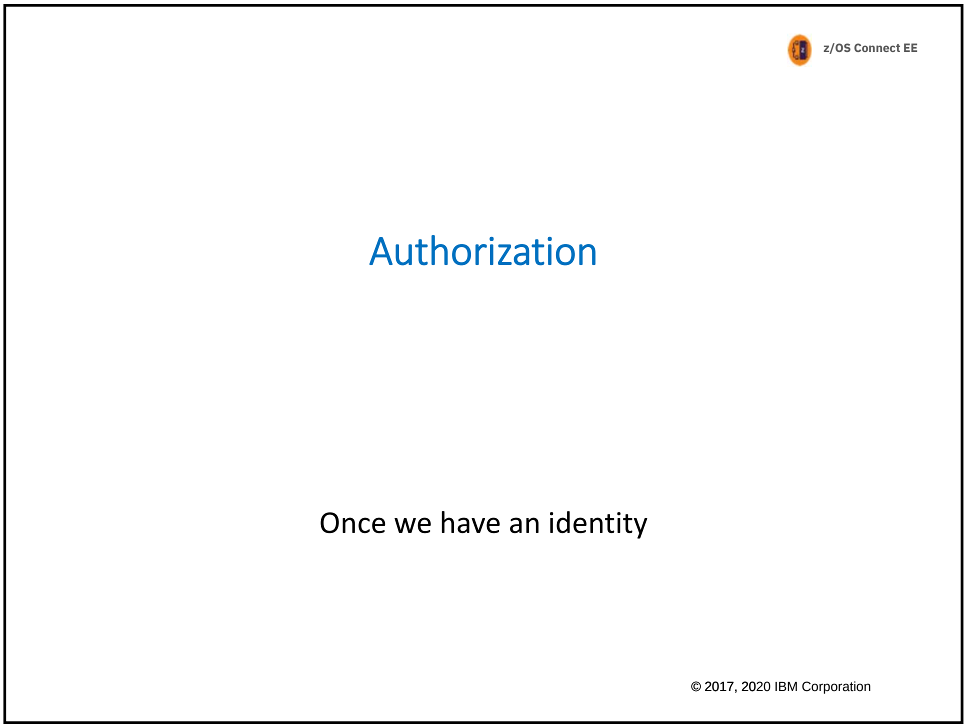
```
<openidConnectClient id="RPssl" inboundPropagation="required"
     signatureAlgorithm="RS256" trustAliasName="JWT-Signer"
     trustStoreRef="jwtTrustStore"
      userIdentityToCreateSubject="sub" mapIdentityToRegistryUser=“true"
     issuerIdentifier=“https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl"
     authnSessionDisabled="true" audiences="myZcee"
     authFilterRef=“JwtAuthFilter”/>
<authFilter id=“API Gateway">
     <remoteAddress id=“ApiAddress" ip="10.7.1.*" matchType="equals"/>
</authFilter>
<authFilter id=“PhoneBook">
      <requestUrl id=“URL"  urlPattern=“/phoneBook/*" matchType="equals"/> </authFilter>
<authFilter id="JwtAuthFilter" >
  <requestHeader id="authHeader" name="Authorization" value="Bearer" matchType="contains"/>
</authFilter>
```

Some alternative filter types

- A ***remoteAddress*** element is compared against the TCP/IP address of the client that sent the request.
- The ***host*** element is compared against the "Host" HTTP request header, which identifies the target host name of the request.
- The ***requestUrl*** element is compared against the URL that is used by the client application to make the request.

29

29

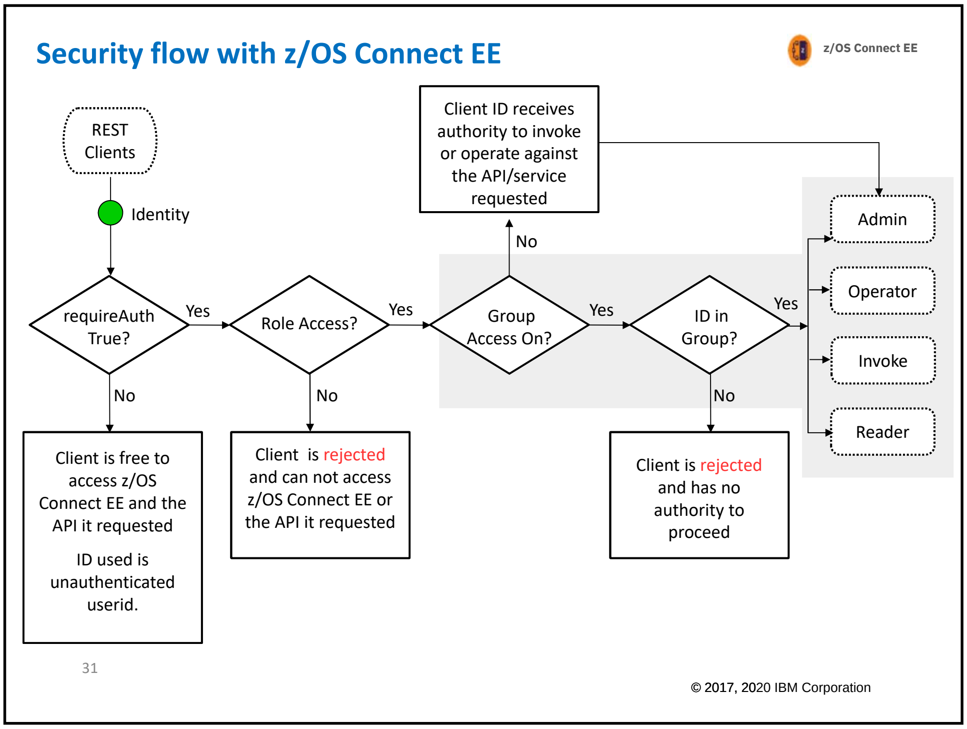


z/OS Connect EE

Authorization

Once we have an identity

30



Security flow with z/OS Connect EE

z/OS Connect EE

REST Clients

Identity

requireAuth True?

Yes

No

Client is free to access z/OS Connect EE and the API it requested

ID used is unauthenticated userid.

Role Access?

Yes

No

Client is rejected and can not access z/OS Connect EE or the API it requested

Group Access On?

No

Client ID receives authority to invoke or operate against the API/service requested

Yes

ID in Group?

No

Client is rejected and has no authority to proceed

Yes

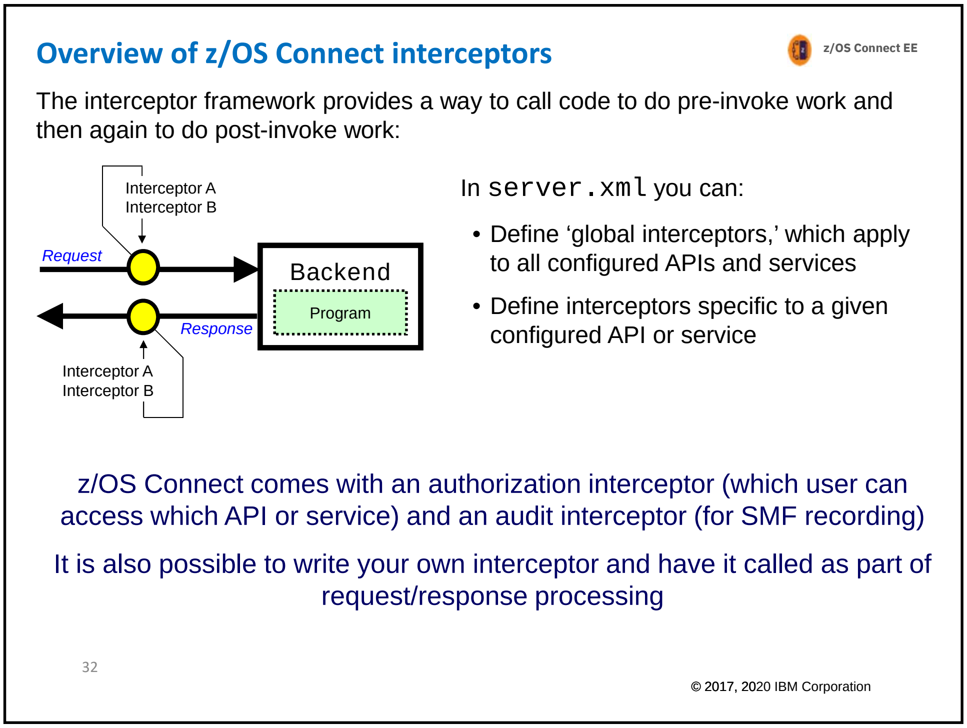
Admin

Operator

Invoke

Reader

31



Overview of z/OS Connect interceptors

z/OS Connect EE

The interceptor framework provides a way to call code to do pre-invoke work and then again to do post-invoke work:

Interceptor A
Interceptor B

Request

Backend

Program

Response

Interceptor A
Interceptor B

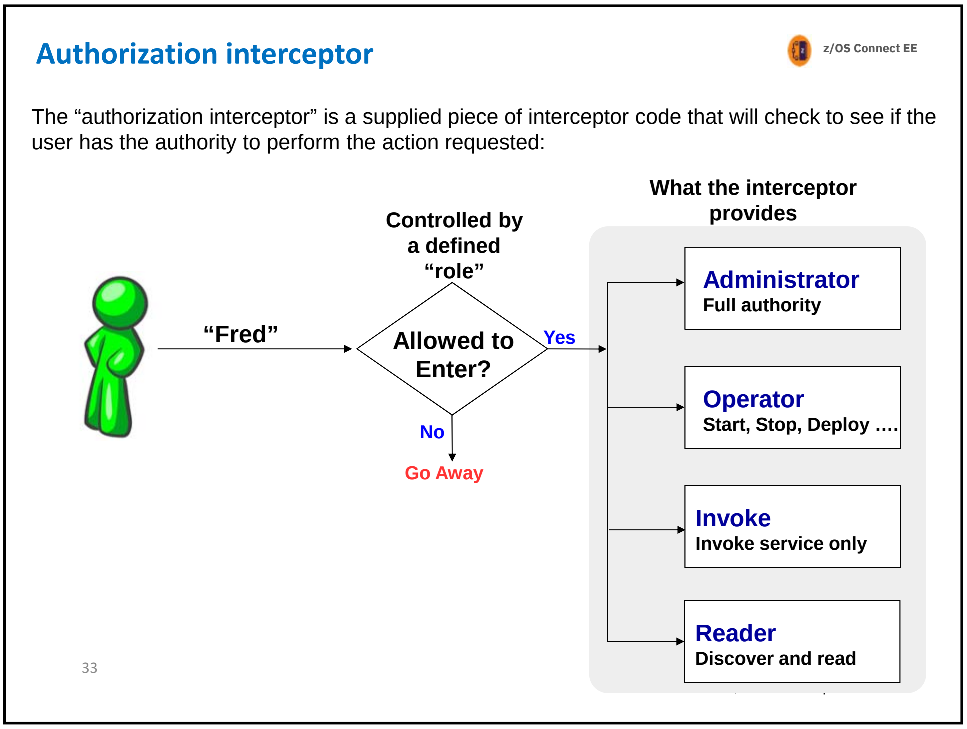
In `server.xml` you can:

- Define ‘global interceptors,’ which apply to all configured APIs and services
- Define interceptors specific to a given configured API or service

z/OS Connect comes with an authorization interceptor (which user can access which API or service) and an audit interceptor (for SMF recording)

It is also possible to write your own interceptor and have it called as part of request/response processing

32



Authorization interceptor

z/OS Connect EE

The “authorization interceptor” is a supplied piece of interceptor code that will check to see if the user has the authority to perform the action requested:

Controlled by a defined “role”

What the interceptor provides

“Fred”

Allowed to Enter?

Yes

No

Go Away

Administrator
Full authority

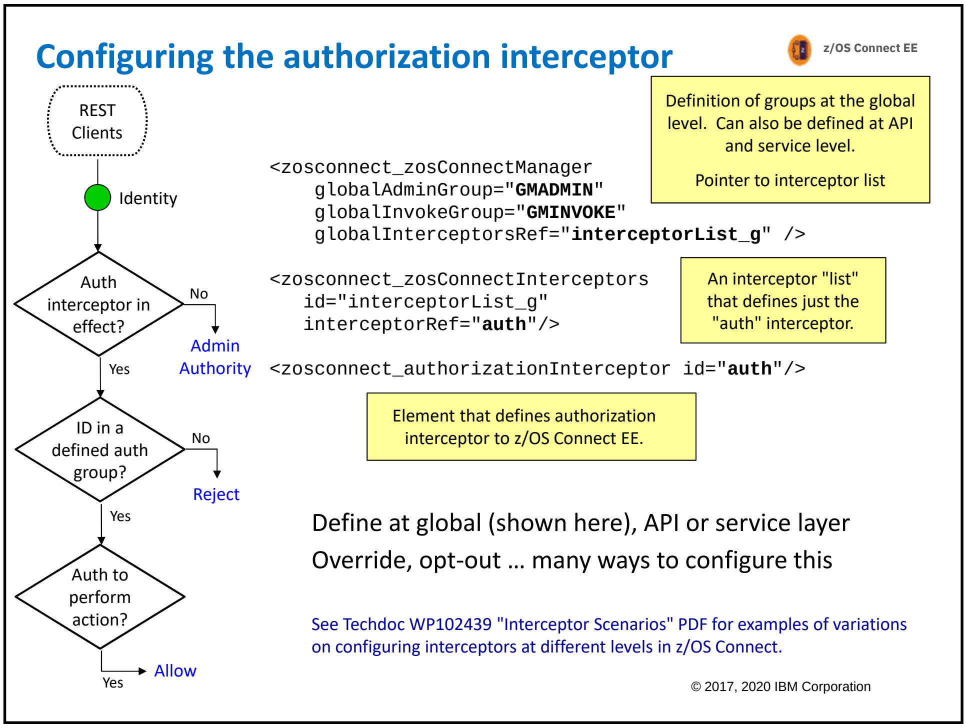
Operator
Start, Stop, Deploy ….

Invoke
Invoke service only

Reader
Discover and read

33

33



Configuring the authorization interceptor

z/OS Connect EE

REST Clients

Identity

Auth interceptor in effect?

No

Admin Authority

Yes

ID in a defined auth group?

No

Reject

Yes

Auth to perform action?

Yes

Allow

Definition of groups at the global level. Can also be defined at API and service level.

Pointer to interceptor list

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

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

<zosconnect_authorizationInterceptor id="auth"/>
```

An interceptor "list" that defines just the "auth" interceptor.

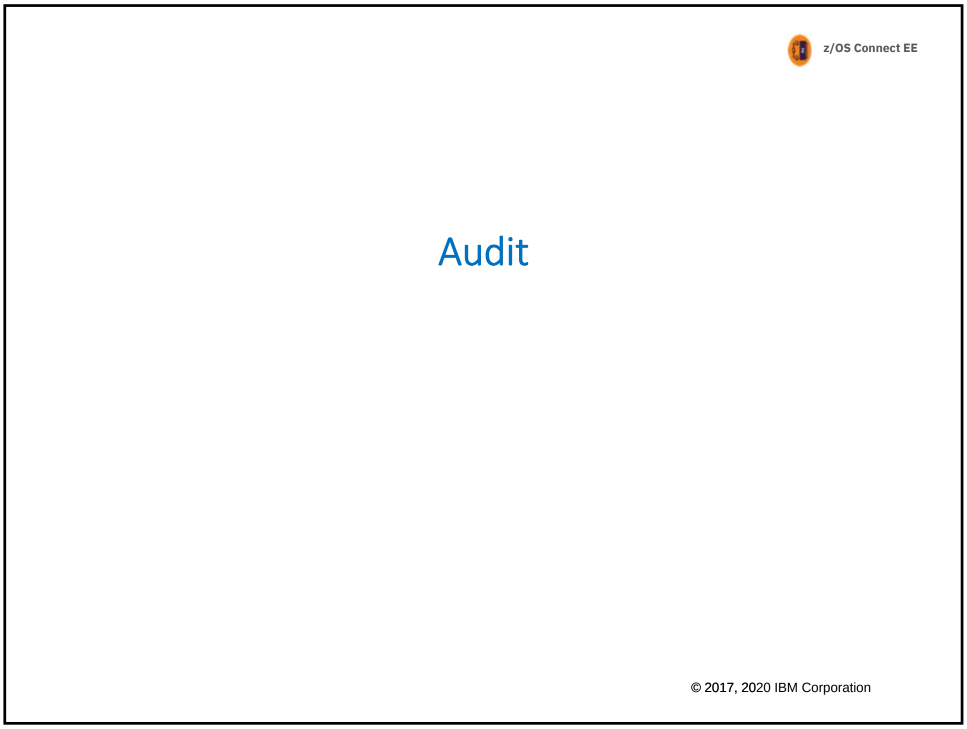
Element that defines authorization interceptor to z/OS Connect EE.

Define at global (shown here), API or service layer

Override, opt-out … many ways to configure this

See Techdoc WP102439 "Interceptor Scenarios" PDF for examples of variations on configuring interceptors at different levels in z/OS Connect.

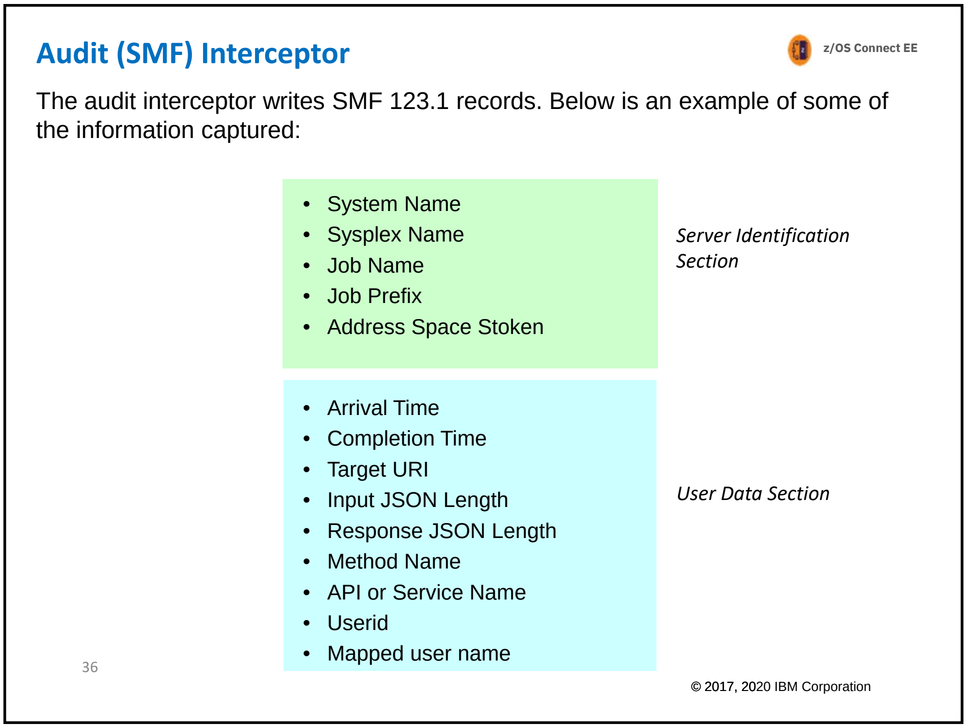
34



z/OS Connect EE

Audit

35



Audit (SMF) Interceptor

z/OS Connect EE

The audit interceptor writes SMF 123.1 records. Below is an example of some of the information captured:

- System Name
- Sysplex Name
- Job Name
- Job Prefix
- Address Space Stoken

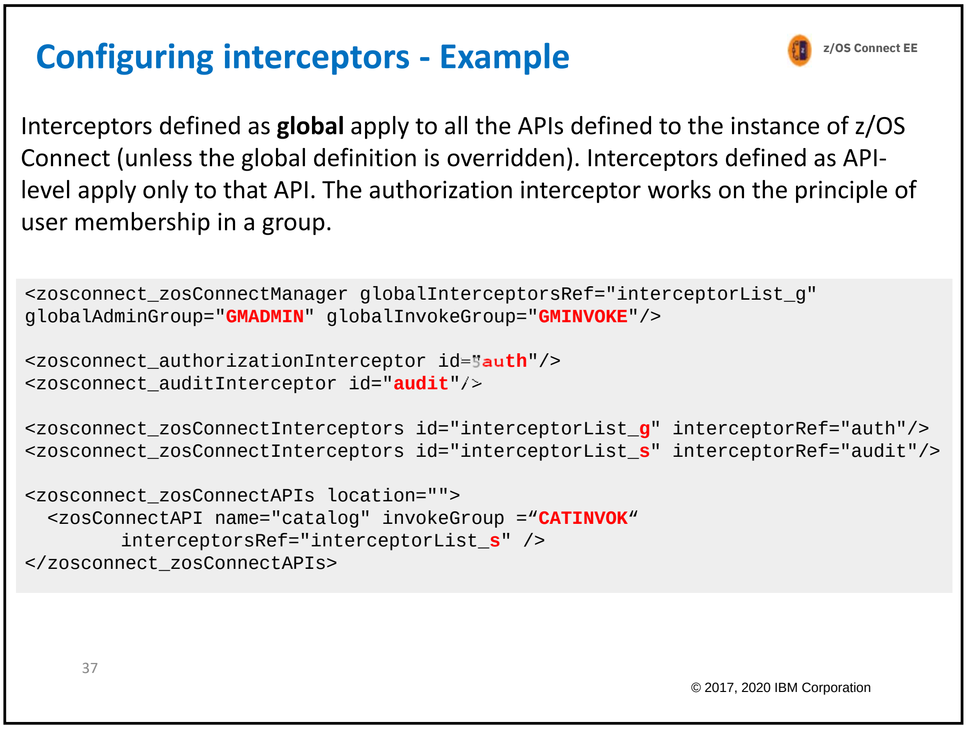
Server Identification Section

- Arrival Time
- Completion Time
- Target URI
- Input JSON Length
- Response JSON Length
- Method Name
- API or Service Name
- Userid
- Mapped user name

User Data Section

36

36



Configuring interceptors - Example

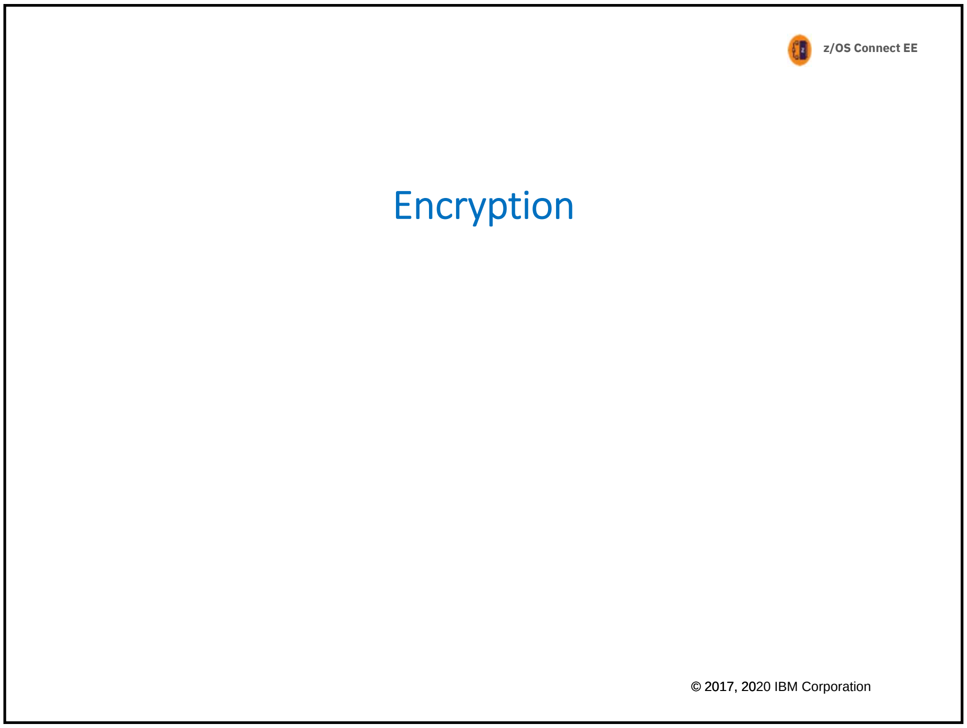
z/OS Connect EE

Interceptors defined as **global** apply to all the APIs defined to the instance of z/OS Connect (unless the global definition is overridden). Interceptors defined as API-level apply only to that API. The authorization interceptor works on the principle of user membership in a group.

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

<zosconnect_authorizationInterceptor id="auth"/>
<zosconnect_auditInterceptor id="audit"/>

<zosconnect_zosConnectInterceptors id="interceptorList_g" interceptorRef="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_s" interceptorRef="audit"/>

<zosconnect_zosConnectAPIs location="">
  <zosConnectAPI name="catalog" invokeGroup =“CATINVOK“
         interceptorsRef="interceptorList_s" />
</zosconnect_zosConnectAPIs>
```

37

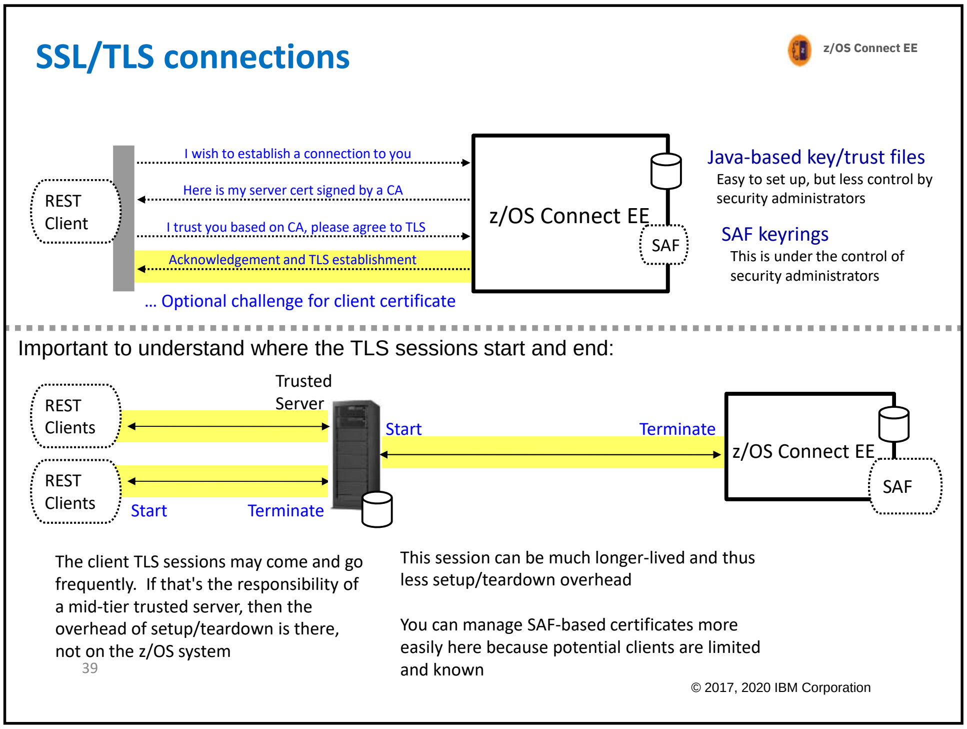
37



z/OS Connect EE

Encryption

38



SSL/TLS connections

z/OS Connect EE
REST Client
I wish to establish a connection to you
Here is my server cert signed by a CA
I trust you based on CA, please agree to TLS
Acknowledgement and TLS establishment
z/OS Connect EE
SAF
… Optional challenge for client certificate

Java-based key/trust files
Easy to set up, but less control by security administrators

SAF keyrings
This is under the control of security administrators

Important to understand where the TLS sessions start and end:

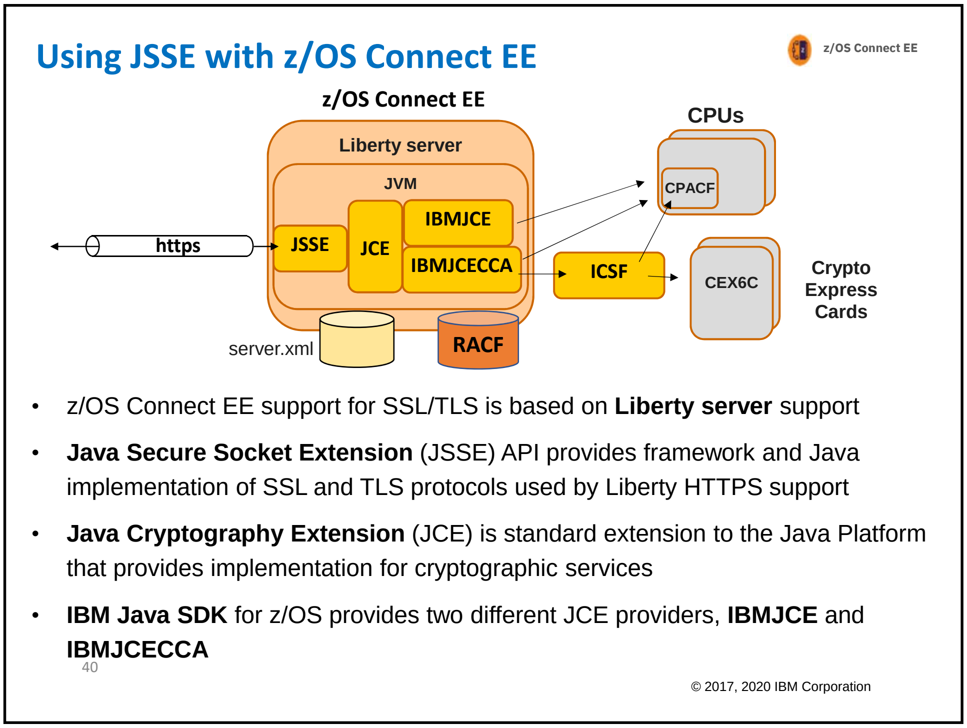
REST Clients
REST Clients
Trusted Server
Start
Terminate
Start
Terminate
z/OS Connect EE
SAF

The client TLS sessions may come and go frequently. If that's the responsibility of a mid-tier trusted server, then the overhead of setup/teardown is there, not on the z/OS system

This session can be much longer-lived and thus less setup/teardown overhead

You can manage SAF-based certificates more easily here because potential clients are limited and known

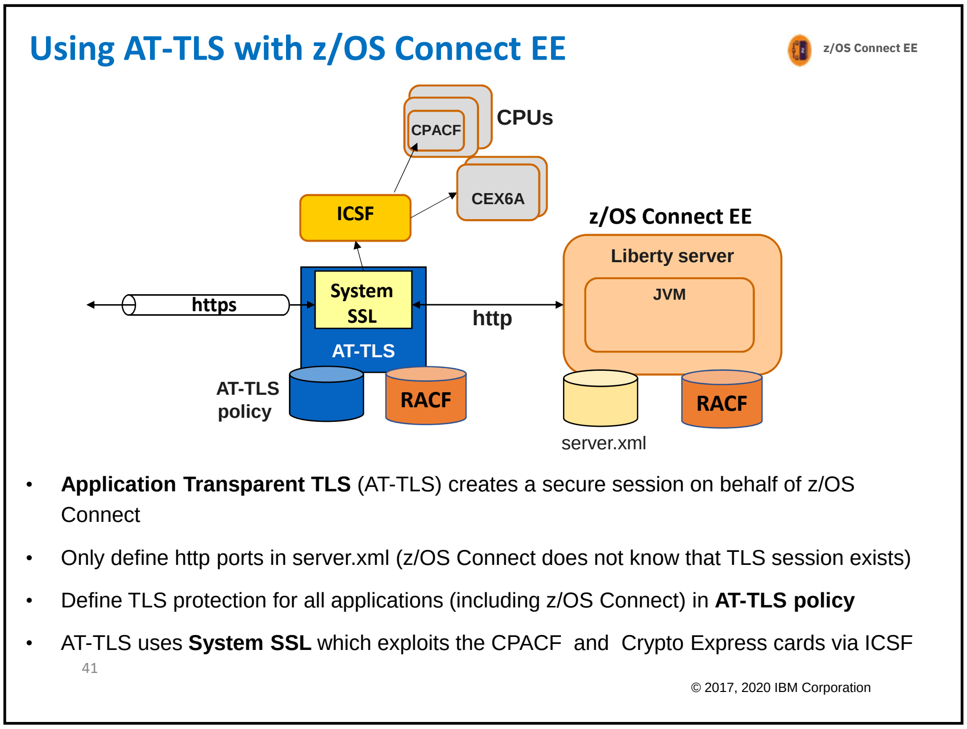
39



Using JSSE with z/OS Connect EE

- z/OS Connect EE support for SSL/TLS is based on **Liberty server** support
- **Java Secure Socket Extension** (JSSE) API provides framework and Java implementation of SSL and TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension** (JCE) is standard extension to the Java Platform that provides implementation for cryptographic services
- **IBM Java SDK** for z/OS provides two different JCE providers, **IBMJCE** and **IBMJCECCA**

40

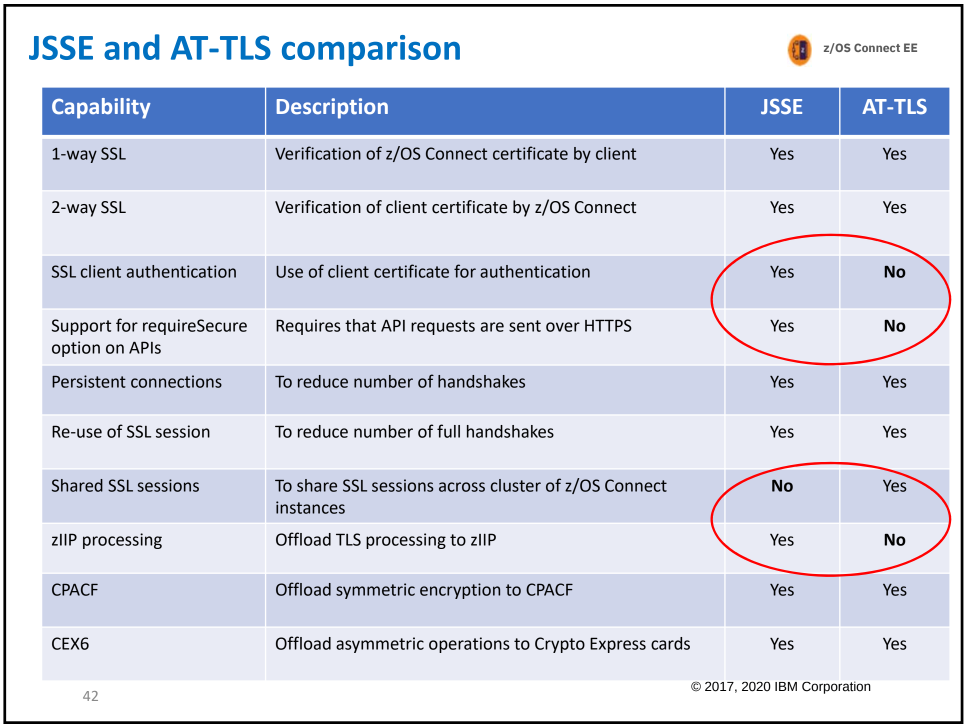


Using AT-TLS with z/OS Connect EE

- **Application Transparent TLS** (AT-TLS) creates a secure session on behalf of z/OS Connect
- Only define http ports in server.xml (z/OS Connect does not know that TLS session exists)
- Define TLS protection for all applications (including z/OS Connect) in **AT-TLS policy**
- AT-TLS uses **System SSL** which exploits the CPACF and Crypto Express cards via ICSF

41

41



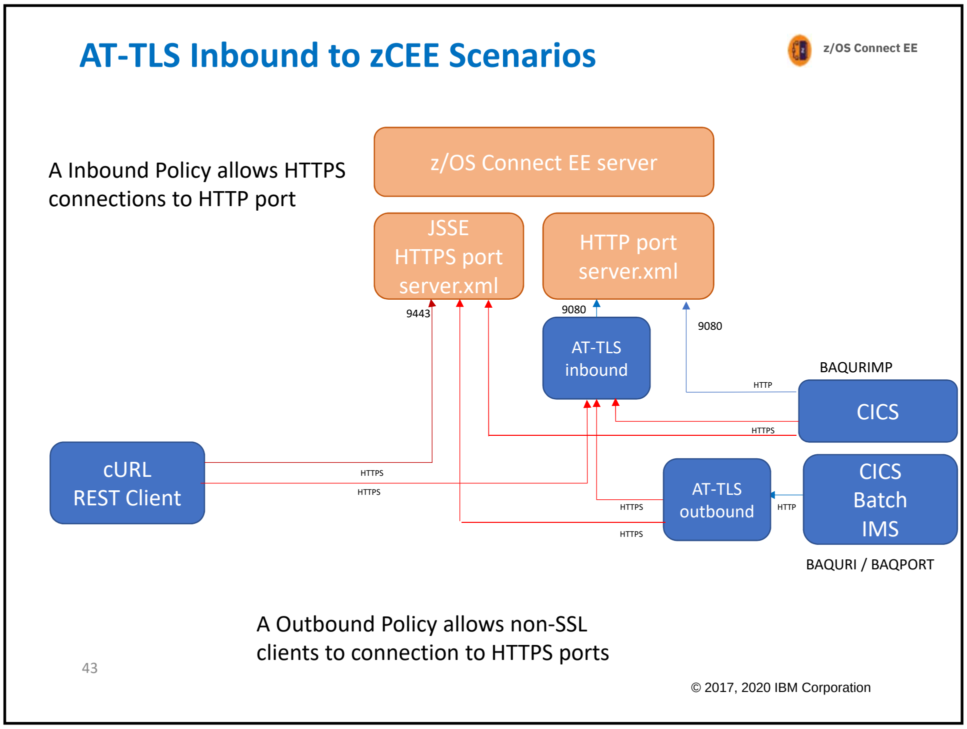
JSSE and AT-TLS comparison

z/OS Connect EE

Capability	Description	JSSE	AT-TLS
1-way SSL	Verification of z/OS Connect certificate by client	Yes	Yes
2-way SSL	Verification of client certificate by z/OS Connect	Yes	Yes
SSL client authentication	Use of client certificate for authentication	Yes	**No**
Support for requireSecure option on APIs	Requires that API requests are sent over HTTPS	Yes	**No**
Persistent connections	To reduce number of handshakes	Yes	Yes
Re-use of SSL session	To reduce number of full handshakes	Yes	Yes
Shared SSL sessions	To share SSL sessions across cluster of z/OS Connect instances	**No**	Yes
zIIP processing	Offload TLS processing to zIIP	Yes	**No**
CPACF	Offload symmetric encryption to CPACF	Yes	Yes
CEX6	Offload asymmetric operations to Crypto Express cards	Yes	Yes

42

42



AT-TLS Inbound to zCEE Scenarios

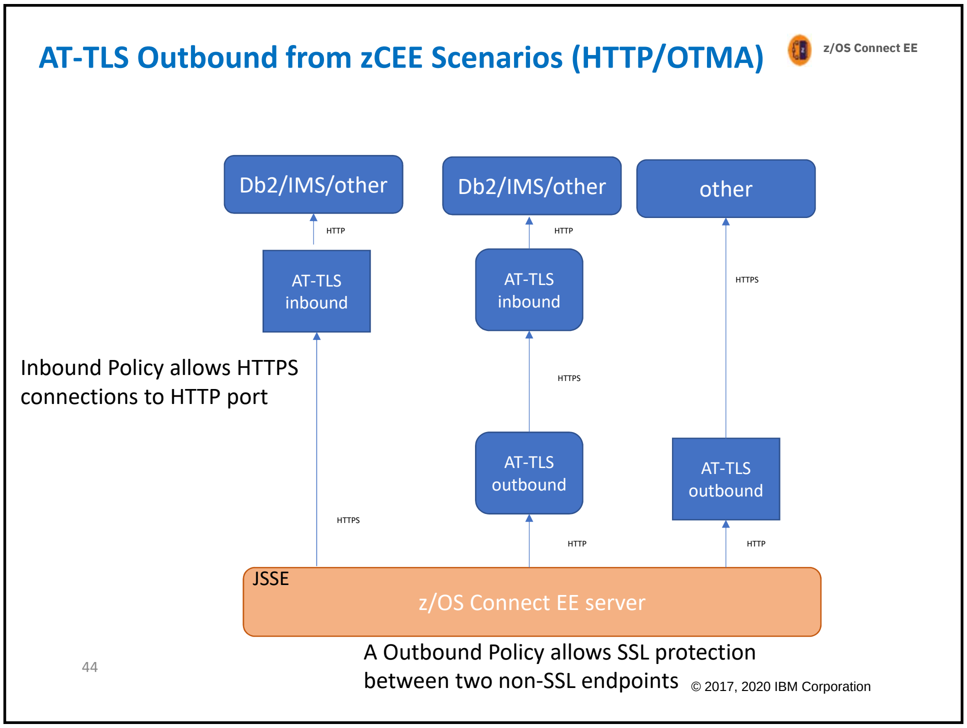
z/OS Connect EE

A Inbound Policy allows HTTPS connections to HTTP port

A Outbound Policy allows non-SSL clients to connection to HTTPS ports

43

43



AT-TLS Outbound from zCEE Scenarios (HTTP/OTMA)

z/OS Connect EE

Inbound Policy allows HTTPS connections to HTTP port

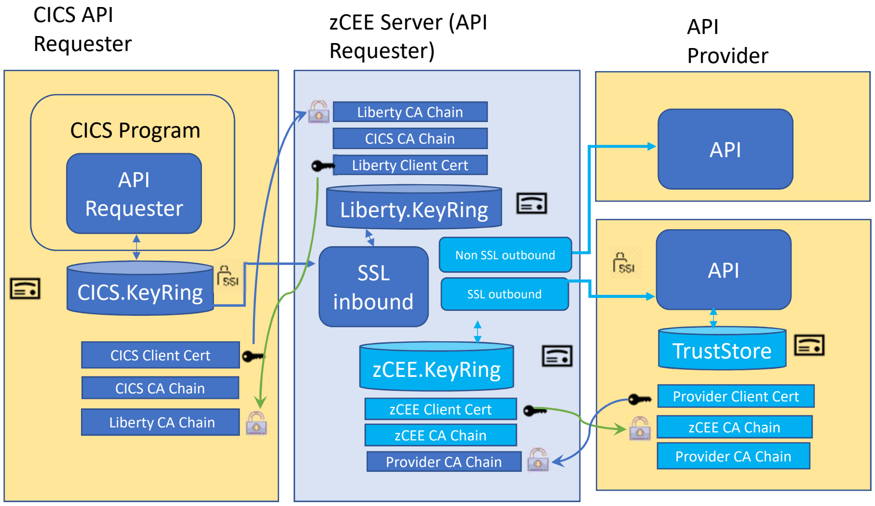
A Outbound Policy allows SSL protection between two non-SSL endpoints

44

44

TLS Handshake Flow

z/OS Connect EE



45

© 2017, 2020 IBM Corporation

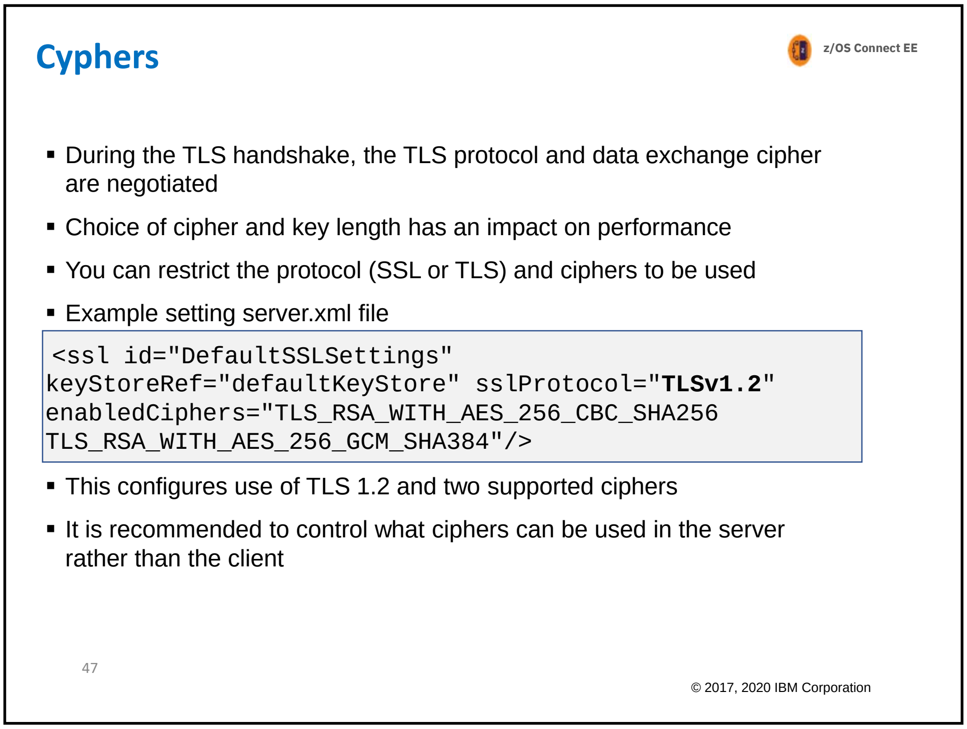
45

z/OS Connect EE

Configuring TLS Encryption with JSSE

© 2017, 2020 IBM Corporation

46



Cyphers

z/OS Connect EE

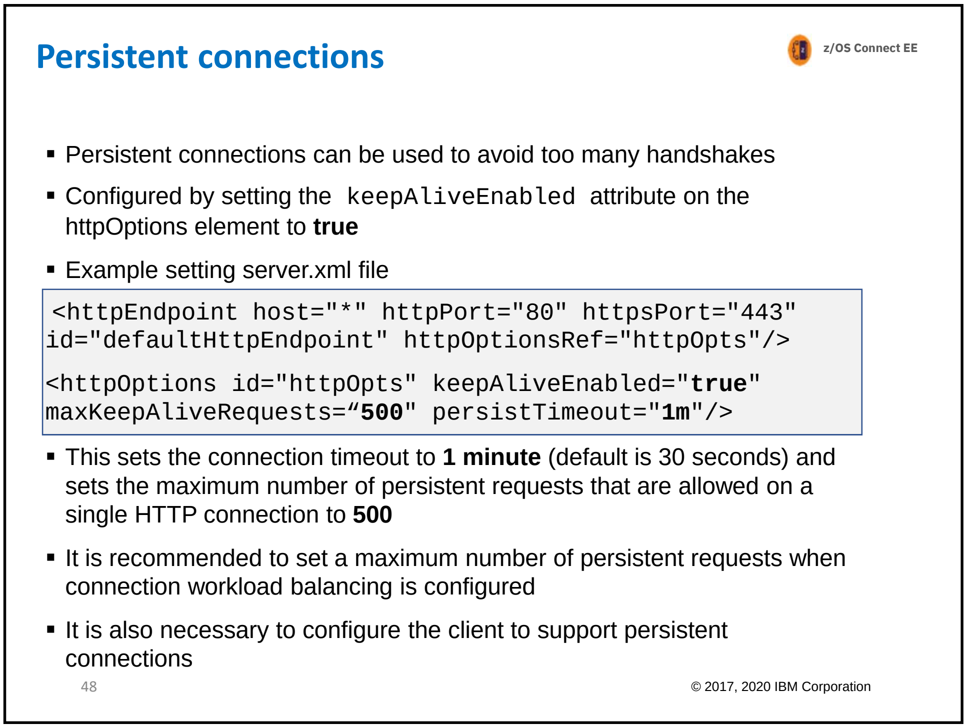
- During the TLS handshake, the TLS protocol and data exchange cipher are negotiated
- Choice of cipher and key length has an impact on performance
- You can restrict the protocol (SSL or TLS) and ciphers to be used
- Example setting server.xml file

```
 <ssl id="DefaultSSLSettings"
keyStoreRef="defaultKeyStore" sslProtocol="TLSv1.2"
enabledCiphers="TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384"/>
```

- This configures use of TLS 1.2 and two supported ciphers
- It is recommended to control what ciphers can be used in the server rather than the client

47

47



Persistent connections

z/OS Connect EE

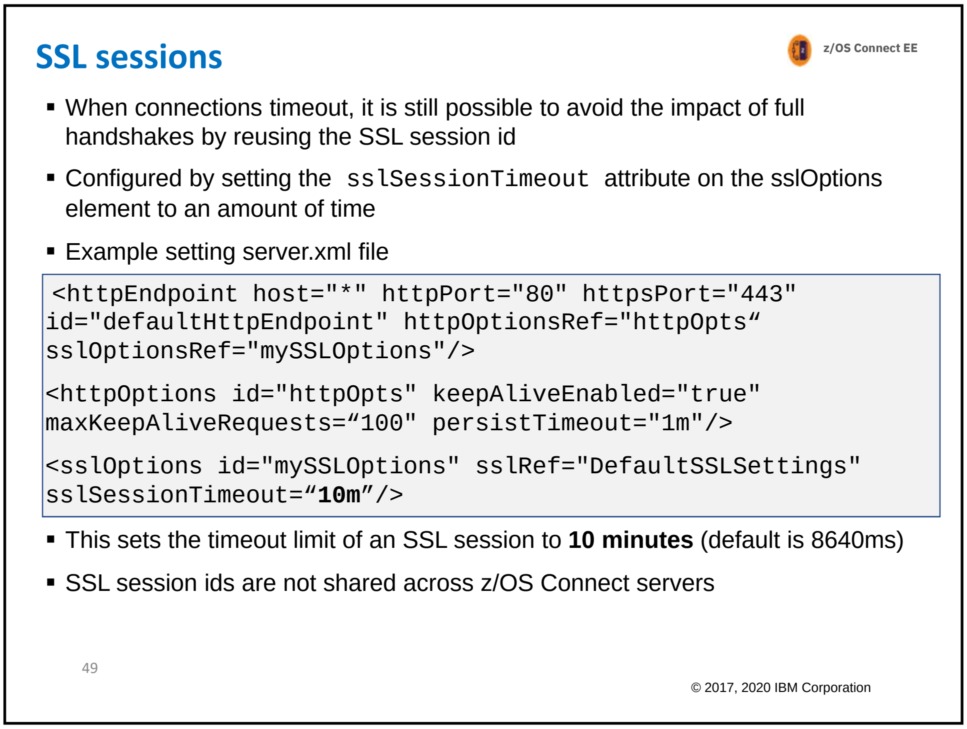
- Persistent connections can be used to avoid too many handshakes
- Configured by setting the `keepAliveEnabled` attribute on the httpOptions element to **true**
- Example setting server.xml file

```
 <httpEndpoint host="*" httpPort="80" httpsPort="443"
id="defaultHttpEndpoint" httpOptionsRef="httpOpts"/>

<httpOptions id="httpOpts" keepAliveEnabled="true"
maxKeepAliveRequests=“500" persistTimeout="1m"/>
```

- This sets the connection timeout to **1 minute** (default is 30 seconds) and sets the maximum number of persistent requests that are allowed on a single HTTP connection to **500**
- It is recommended to set a maximum number of persistent requests when connection workload balancing is configured
- It is also necessary to configure the client to support persistent connections

48

48



SSL sessions

z/OS Connect EE

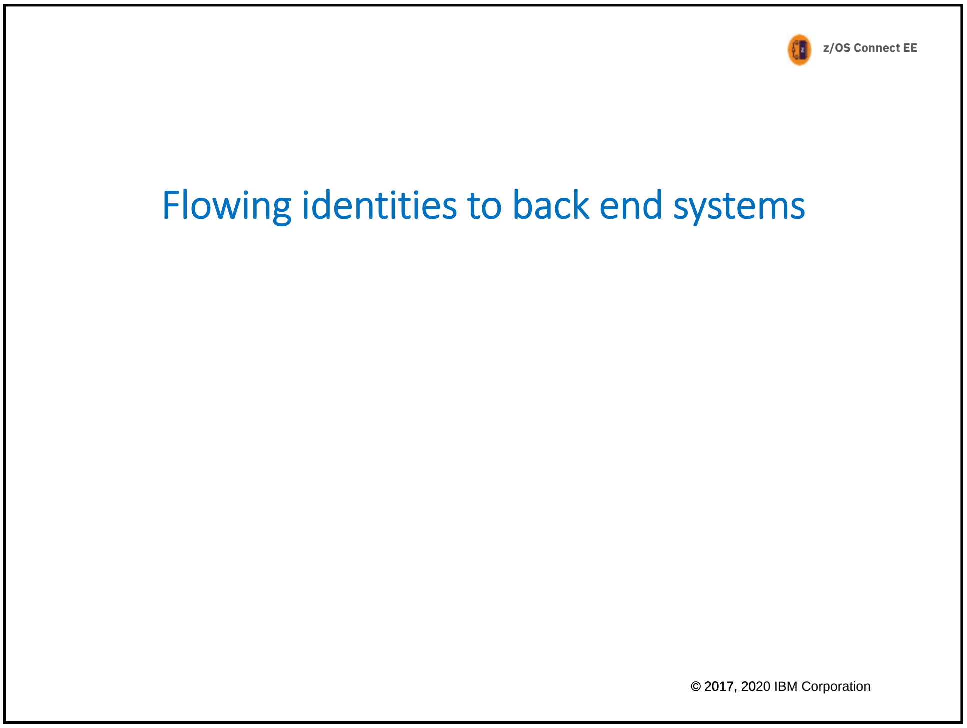
- When connections timeout, it is still possible to avoid the impact of full handshakes by reusing the SSL session id
- Configured by setting the `sslSessionTimeout` attribute on the sslOptions element to an amount of time
- Example setting server.xml file

```
 <httpEndpoint host="*" httpPort="80" httpsPort="443"
id="defaultHttpEndpoint" httpOptionsRef="httpOpts“
sslOptionsRef="mySSLOptions"/>

<httpOptions id="httpOpts" keepAliveEnabled="true"
maxKeepAliveRequests=“100" persistTimeout="1m"/>

<sslOptions id="mySSLOptions" sslRef="DefaultSSLSettings"
sslSessionTimeout=“10m”/>
```

- This sets the timeout limit of an SSL session to **10 minutes** (default is 8640ms)
- SSL session ids are not shared across z/OS Connect servers

49

49



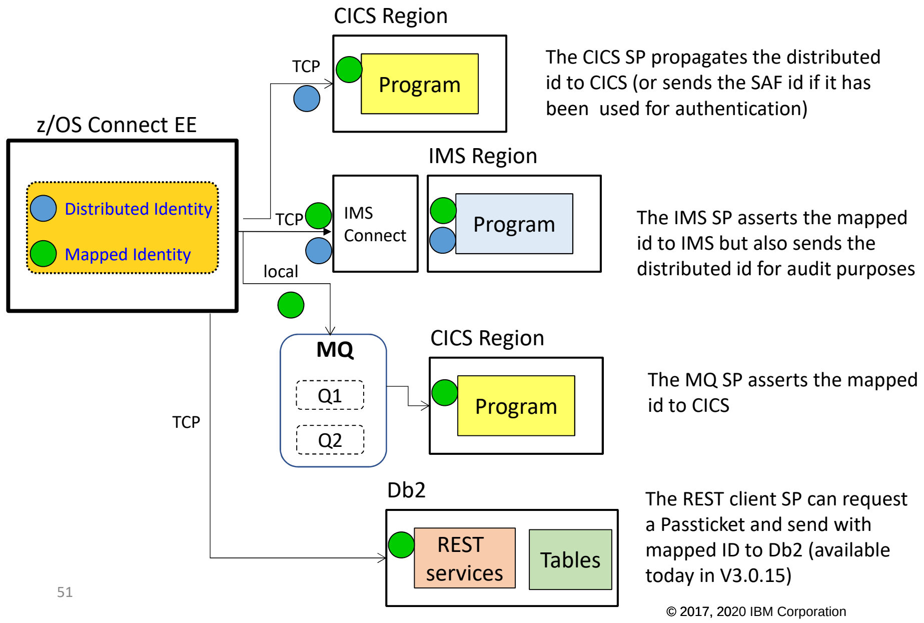
z/OS Connect EE

Flowing identities to back end systems

50

Flowing an identity to the back end

z/OS Connect EE



The CICS SP propagates the distributed id to CICS (or sends the SAF id if it has been used for authentication)

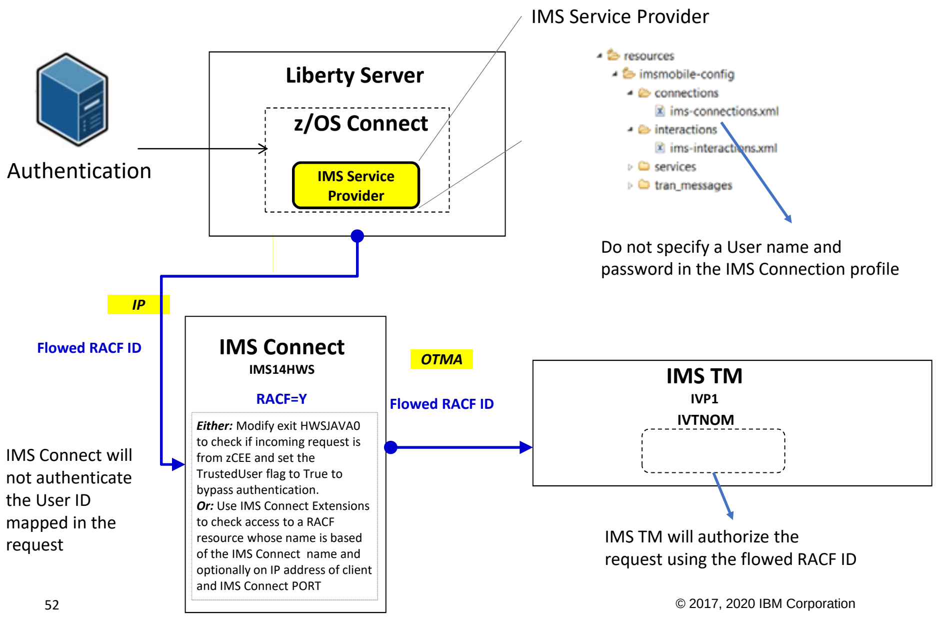
The IMS SP asserts the mapped id to IMS but also sends the distributed id for audit purposes

The MQ SP asserts the mapped id to CICS

The REST client SP can request a Passticket and send with mapped ID to Db2 (available today in V3.0.15)

Flowing a RACF ID to IMS

z/OS Connect EE



Do not specify a User name and password in the IMS Connection profile

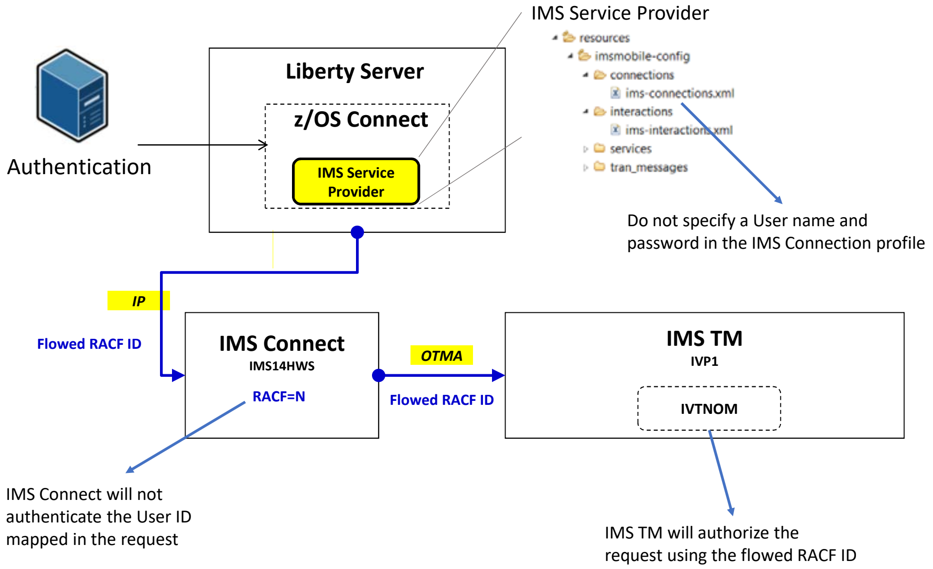
Either: Modify exit HWSJAVA0 to check if incoming request is from zCEE and set the TrustedUser flag to True to bypass authentication.
Or: Use IMS Connect Extensions to check access to a RACF resource whose name is based of the IMS Connect name and optionally on IP address of client and IMS Connect PORT

IMS Connect will not authenticate the User ID mapped in the request

IMS TM will authorize the request using the flowed RACF ID

Flowing a RACF ID to IMS

z/OS Connect EE



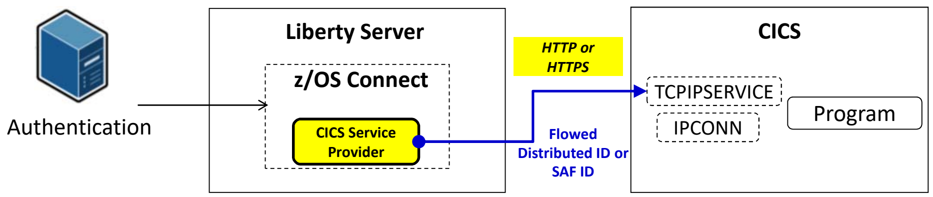
Do not specify a User name and password in the IMS Connection profile

IMS Connect will not authenticate the User ID mapped in the request

IMS TM will authorize the request using the flowed RACF ID

Flowing a user ID with CICS service provider

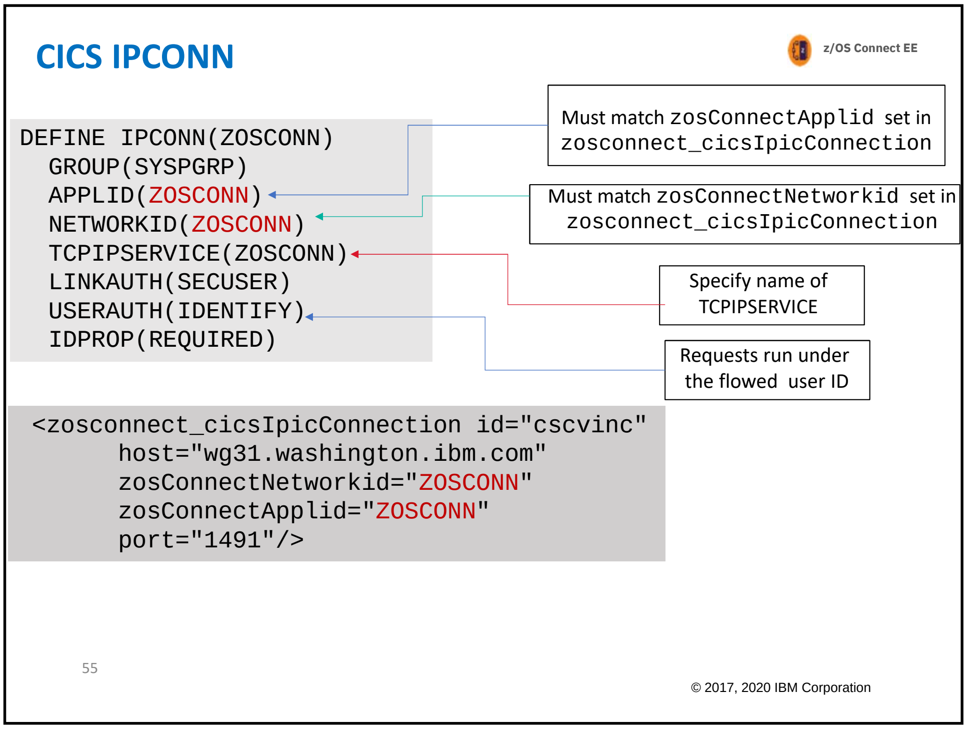
z/OS Connect EE



IPIC connections enforce **bind** security to prevent an unauthorized client system from connecting to CICS, **link** security to restrict the resources that can be accessed over a connection to a CICS system, and **user** security to restrict the CICS resources that can be accessed by a user

Distributed identities can be propagated to CICS and then mapped to a RACF user ID by CICS. You can then view the distinguished name and realm for a distributed identity in the association data of the CICS task. **Important**: If the z/OS Connect EE server is not in the same sysplex as the CICS system, you must use an IPIC SSL connection that is configured with client authentication.

If a SAF ID is used for authentication (e.g basicauth with a SAF registry) then the SAF ID is passed to CICS.



CICS IPCONN

z/OS Connect EE

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

Must match `zosConnectApplid` set in `zosconnect_cicsIpicConnection`

Must match `zosConnectNetworkid` set in `zosconnect_cicsIpicConnection`

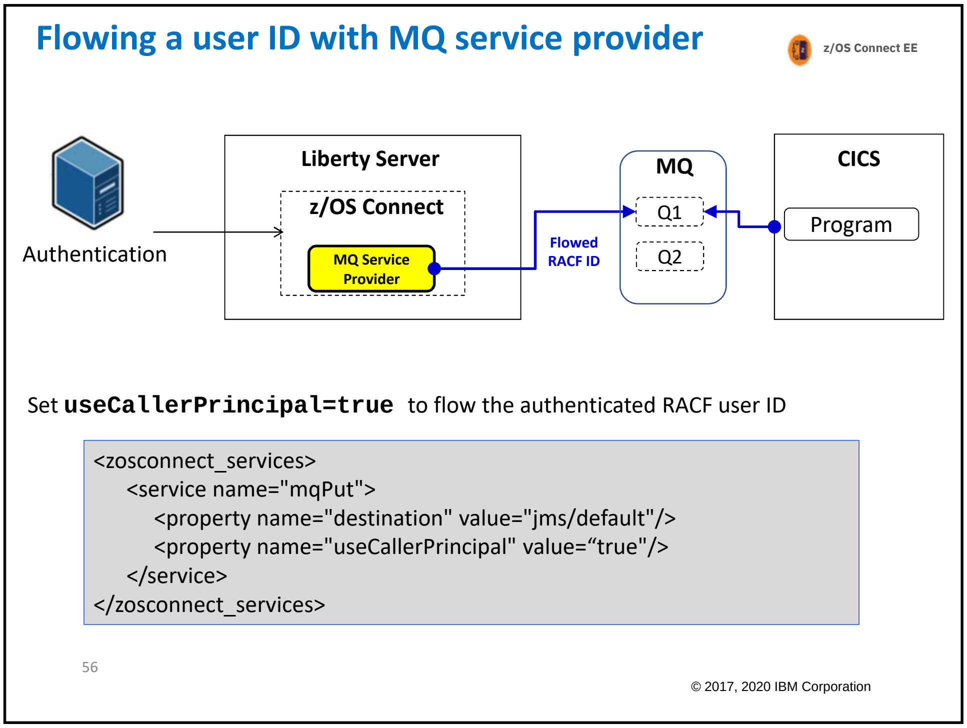
Specify name of TCPIPSERVICE

Requests run under the flowed user ID

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

55

55



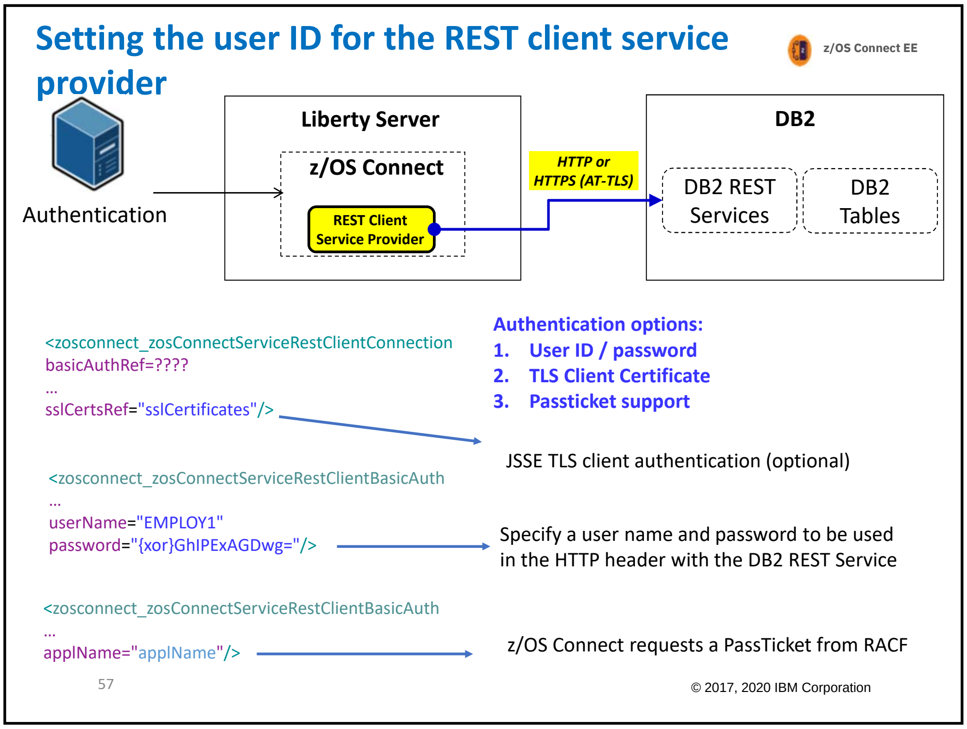
Flowing a user ID with MQ service provider

Set **`useCallerPrincipal=true`** to flow the authenticated RACF user ID

```
<zosconnect_services>
    <service name="mqPut">
        <property name="destination" value="jms/default"/>
        <property name="useCallerPrincipal" value="true"/>
    </service>
</zosconnect_services>
```

56

56



Setting the user ID for the REST client service provider

```
<zosconnect_zosConnectServiceRestClientConnection
basicAuthRef=????
…
sslCertsRef="sslCertificates"/>
```

Authentication options:

1. **User ID / password**
2. **TLS Client Certificate**
3. **Passticket support**

JSSE TLS client authentication (optional)

```
<zosconnect_zosConnectServiceRestClientBasicAuth
…
userName="EMPLOY1"
password="{xor}GhIPExAGDwg="/>
```

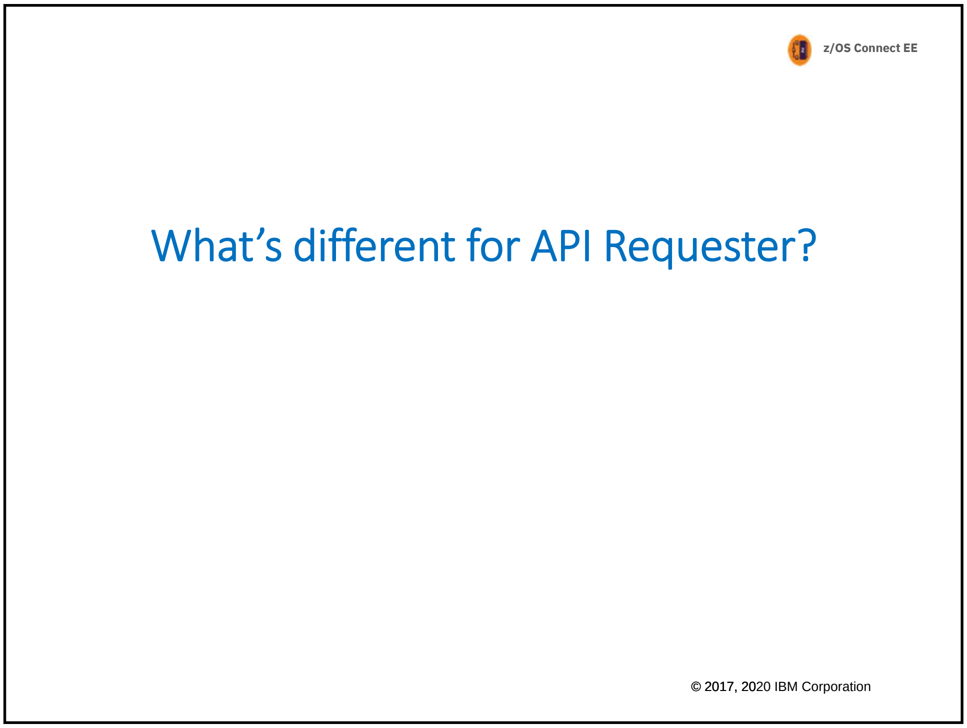
Specify a user name and password to be used in the HTTP header with the DB2 REST Service

```
<zosconnect_zosConnectServiceRestClientBasicAuth
…
applName="applName"/>
```

z/OS Connect requests a PassTicket from RACF

57

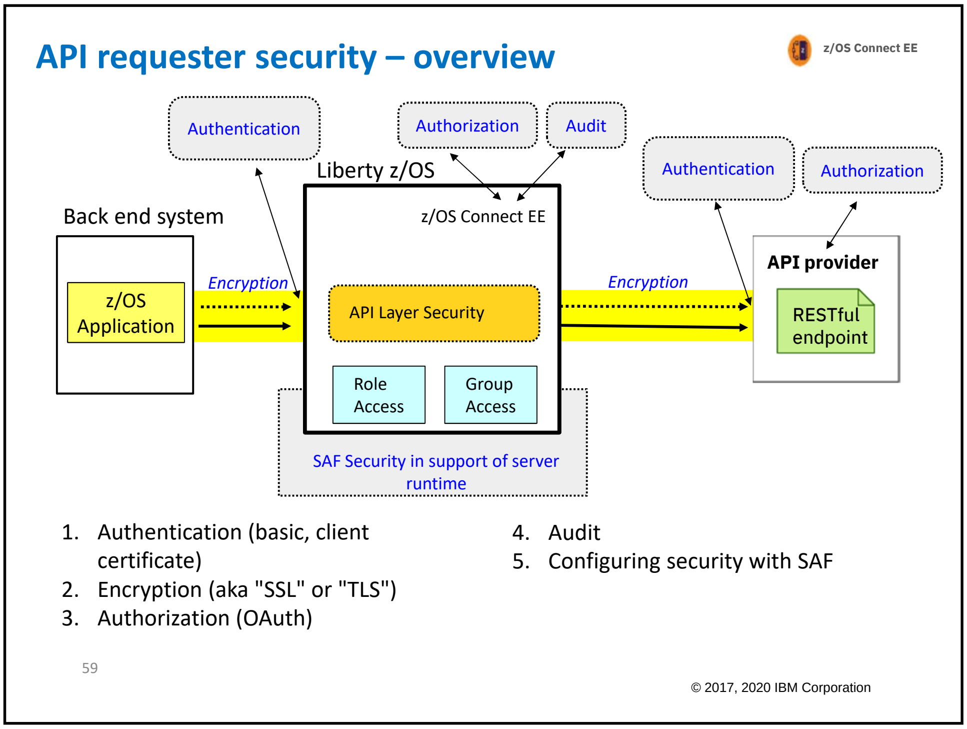
57



z/OS Connect EE

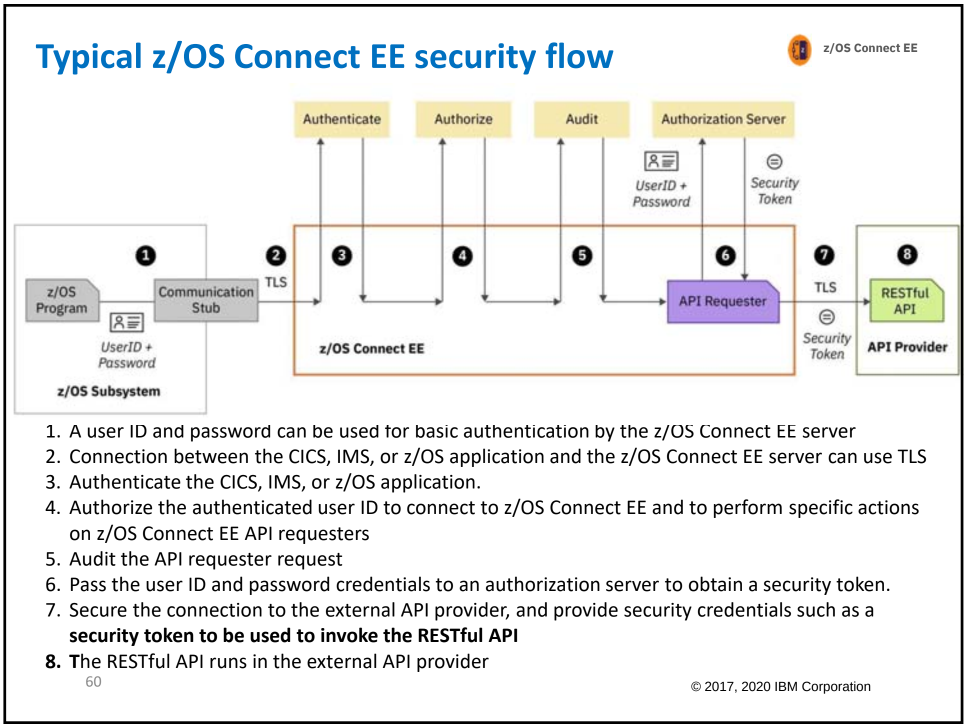
What’s different for API Requester?

58



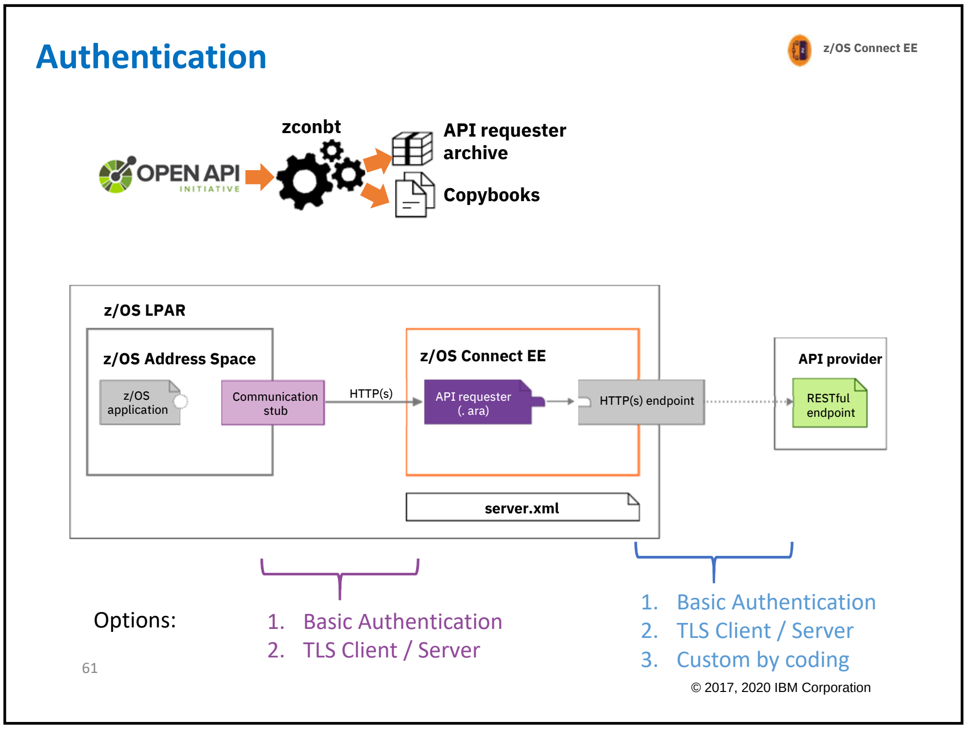
API requester security – overview

1. Authentication (basic, client certificate)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (OAuth)
4. Audit
5. Configuring security with SAF



Typical z/OS Connect EE security flow

1. A user ID and password can be used for basic authentication by the z/OS Connect EE server
2. Connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to z/OS Connect EE and to perform specific actions on z/OS Connect EE API requesters
5. Audit the API requester request
6. Pass the user ID and password credentials to an authorization server to obtain a security token.
7. Secure the connection to the external API provider, and provide security credentials such as a **security token to be used to invoke the RESTful API**
8. **T**he RESTful API runs in the external API provider



Authentication

z/OS Connect EE

OPEN API INITIATIVE

zconbt

API requester archive

Copybooks

z/OS LPAR

z/OS Address Space

z/OS application

Communication stub

HTTP(s)

z/OS Connect EE

API requester (. ara)

HTTP(s) endpoint

server.xml

API provider

RESTful endpoint

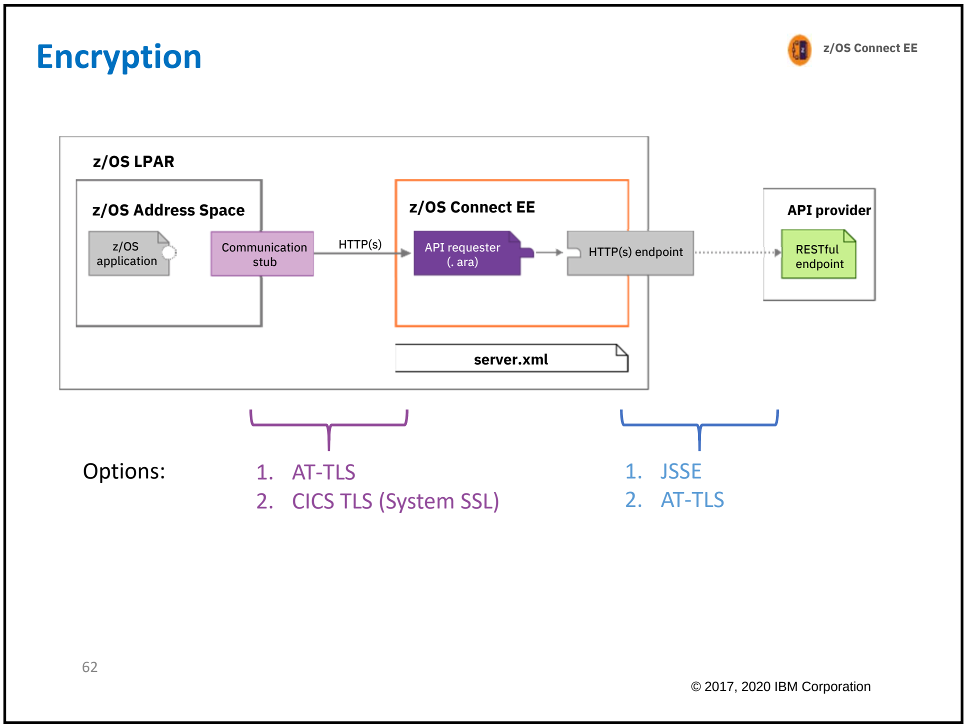
Options:

1. Basic Authentication
2. TLS Client / Server

1. Basic Authentication
2. TLS Client / Server
3. Custom by coding

61

61



Encryption

z/OS Connect EE

z/OS LPAR

z/OS Address Space

z/OS application

Communication stub

HTTP(s)

z/OS Connect EE

API requester (. ara)

HTTP(s) endpoint

server.xml

API provider

RESTful endpoint

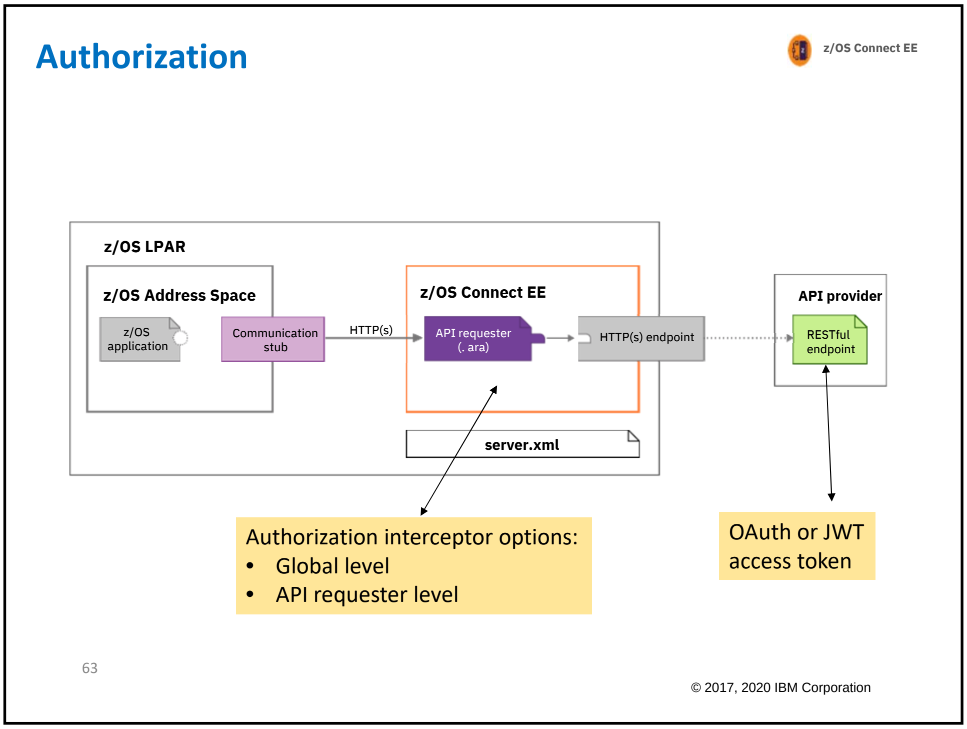
Options:

1. AT-TLS
2. CICS TLS (System SSL)

1. JSSE
2. AT-TLS

62

62



Authorization

z/OS Connect EE

z/OS LPAR

z/OS Address Space

z/OS application

Communication stub

HTTP(s)

z/OS Connect EE

API requester (. ara)

HTTP(s) endpoint

API provider

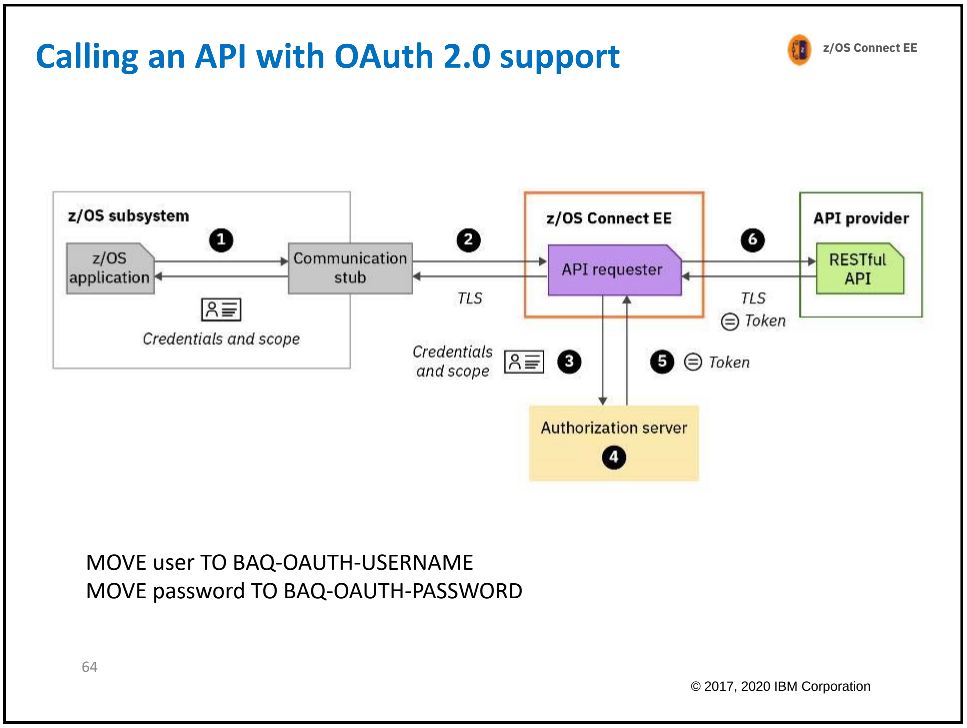
RESTful endpoint

server.xml

Authorization interceptor options:

- Global level
- API requester level

OAuth or JWT access token



Calling an API with OAuth 2.0 support

z/OS Connect EE

z/OS subsystem

z/OS application

❶

Communication stub

Credentials and scope

❷

TLS

z/OS Connect EE

API requester

❻

API provider

RESTful API

TLS

Token

Credentials and scope

❸

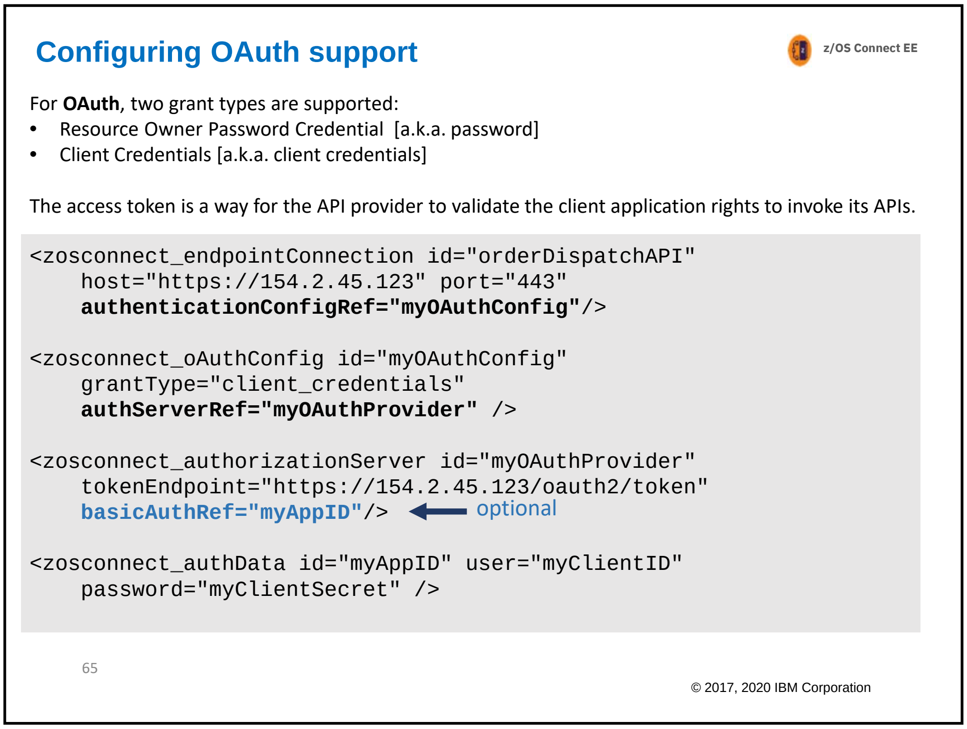
❺

Token

Authorization server

❹

MOVE user TO BAQ-OAUTH-USERNAME
MOVE password TO BAQ-OAUTH-PASSWORD



Configuring OAuth support

z/OS Connect EE

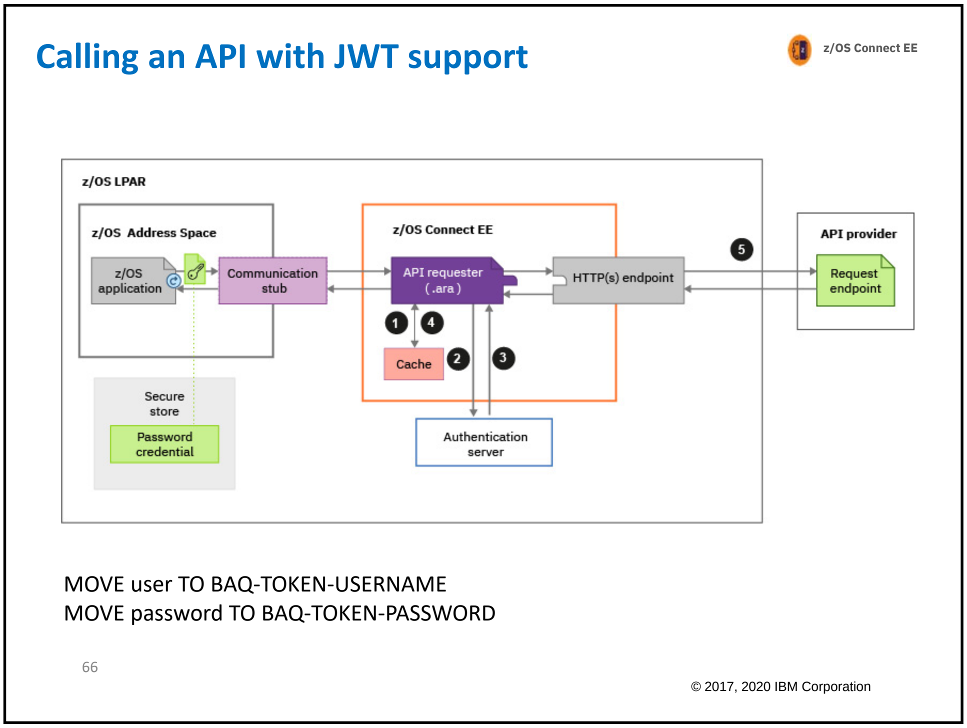
For **OAuth**, two grant types are supported:
- Resource Owner Password Credential [a.k.a. password]
- Client Credentials [a.k.a. client credentials]

The access token is a way for the API provider to validate the client application rights to invoke its APIs.

```
<zosconnect_endpointConnection id="orderDispatchAPI"
    host="https://154.2.45.123" port="443"
    authenticationConfigRef="myOAuthConfig"/>

<zosconnect_oAuthConfig id="myOAuthConfig"
    grantType="client_credentials"
    authServerRef="myOAuthProvider" />

<zosconnect_authorizationServer id="myOAuthProvider"
    tokenEndpoint="https://154.2.45.123/oauth2/token"
    basicAuthRef="myAppID"/>  ← optional

<zosconnect_authData id="myAppID" user="myClientID"
    password="myClientSecret" />
```

65

65



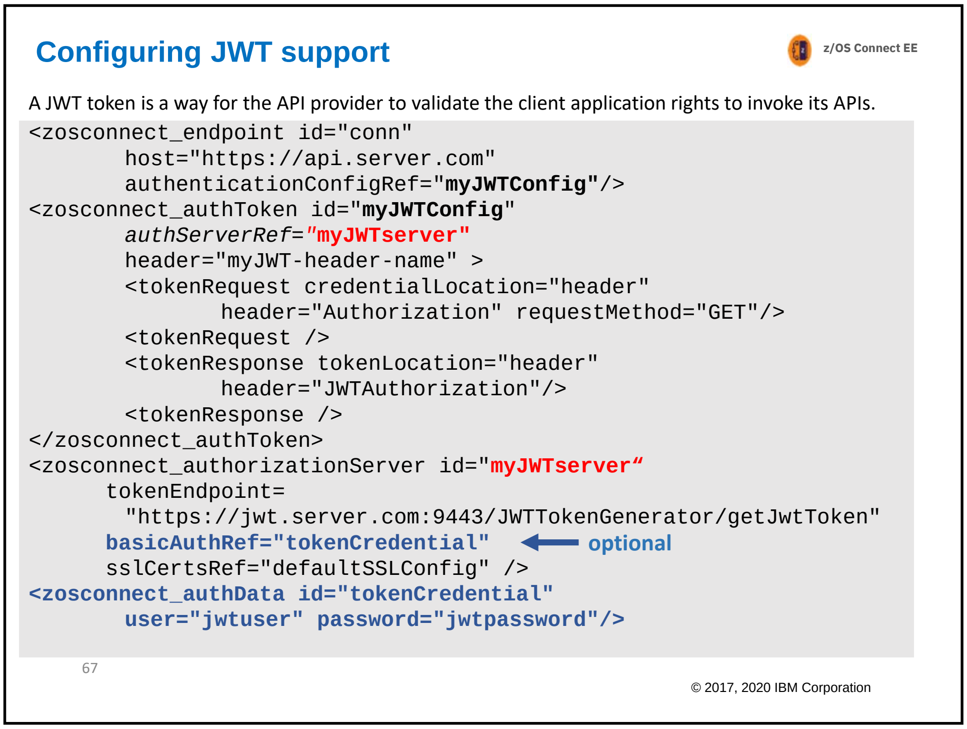
Calling an API with JWT support

z/OS Connect EE

MOVE user TO BAQ-TOKEN-USERNAME
MOVE password TO BAQ-TOKEN-PASSWORD

66

66



Configuring JWT support

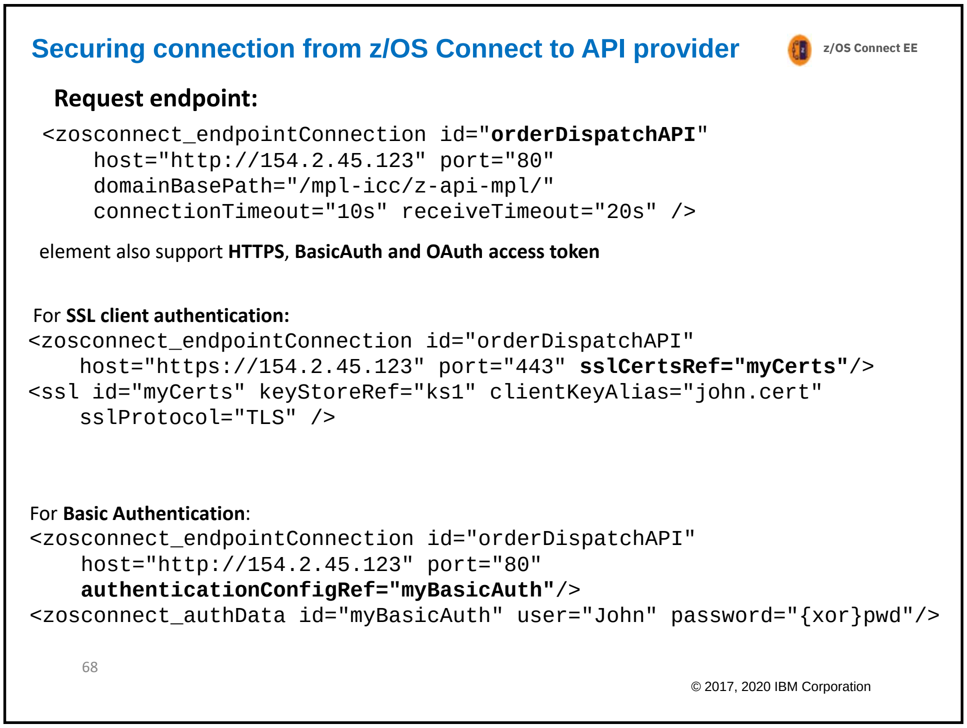
z/OS Connect EE

A JWT token is a way for the API provider to validate the client application rights to invoke its APIs.

```
<zosconnect_endpoint id="conn"
       host="https://api.server.com"
       authenticationConfigRef="myJWTConfig"/>
<zosconnect_authToken id="myJWTConfig"
       authServerRef="myJWTserver"
       header="myJWT-header-name" >
       <tokenRequest credentialLocation="header"
              header="Authorization" requestMethod="GET"/>
       <tokenRequest />
       <tokenResponse tokenLocation="header"
              header="JWTAuthorization"/>
       <tokenResponse />
</zosconnect_authToken>
<zosconnect_authorizationServer id="myJWTserver“
      tokenEndpoint=
       "https://jwt.server.com:9443/JWTTokenGenerator/getJwtToken"
      basicAuthRef="tokenCredential"   ⟵ optional
      sslCertsRef="defaultSSLConfig" />
<zosconnect_authData id="tokenCredential"
       user="jwtuser" password="jwtpassword"/>
```

67

67



Securing connection from z/OS Connect to API provider

z/OS Connect EE

Request endpoint:

```
<zosconnect_endpointConnection id="orderDispatchAPI"
    host="http://154.2.45.123" port="80"
    domainBasePath="/mpl-icc/z-api-mpl/"
    connectionTimeout="10s" receiveTimeout="20s" />
```

element also support **HTTPS**, **BasicAuth and OAuth access token**

For **SSL client authentication:**

```
<zosconnect_endpointConnection id="orderDispatchAPI"
    host="https://154.2.45.123" port="443" sslCertsRef="myCerts"/>
<ssl id="myCerts" keyStoreRef="ks1" clientKeyAlias="john.cert"
    sslProtocol="TLS" />
```

For **Basic Authentication**:

```
<zosconnect_endpointConnection id="orderDispatchAPI"
    host="http://154.2.45.123" port="80"
    authenticationConfigRef="myBasicAuth"/>
<zosconnect_authData id="myBasicAuth" user="John" password="{xor}pwd"/>
```

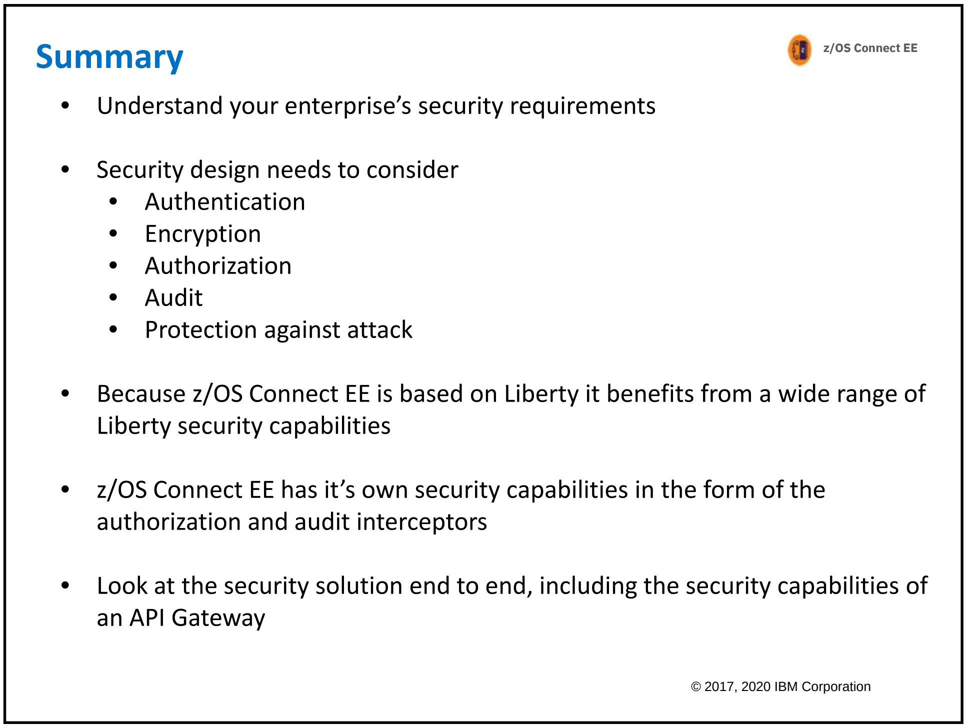
68

68

z/OS Connect EE

Summary

69



Summary

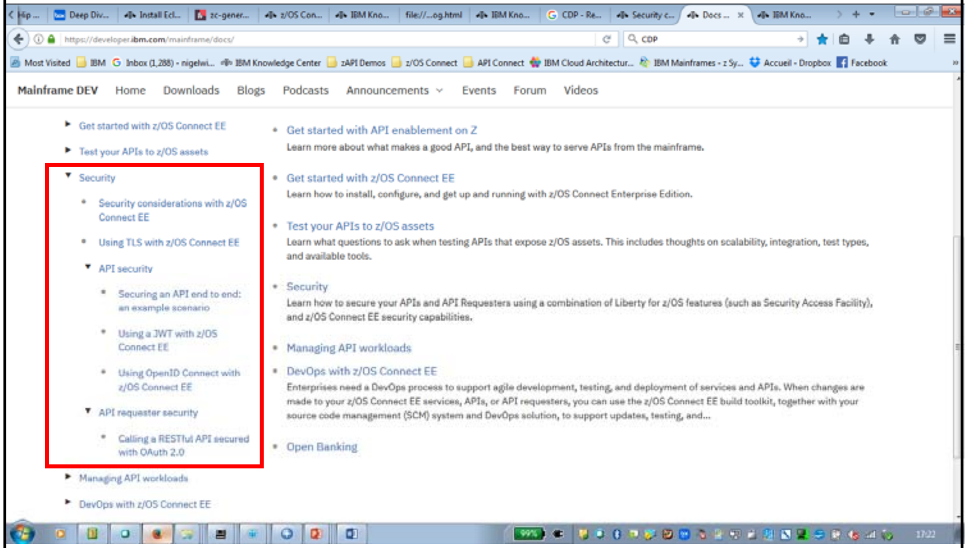
z/OS Connect EE

- Understand your enterprise’s security requirements
- Security design needs to consider
  - Authentication
  - Encryption
  - Authorization
  - Audit
  - Protection against attack
- Because z/OS Connect EE is based on Liberty it benefits from a wide range of Liberty security capabilities
- z/OS Connect EE has it’s own security capabilities in the form of the authorization and audit interceptors
- Look at the security solution end to end, including the security capabilities of an API Gateway

70

More information

z/OS Connect EE



Mainframe DEV   Home   Downloads   Blogs   Podcasts   Announcements   Events   Forum   Videos

- Get started with z/OS Connect EE
- Test your APIs to z/OS assets
- Security
  - Security considerations with z/OS Connect EE
  - Using TLS with z/OS Connect EE
  - API security
    - Securing an API end to end: an example scenario
    - Using a JWT with z/OS Connect EE
    - Using OpenID Connect with z/OS Connect EE
  - API requester security
    - Calling a RESTful API secured with OAuth 2.0
- Managing API workloads
- DevOps with z/OS Connect EE

- Get started with API enablement on Z
  Learn more about what makes a good API, and the best way to serve APIs from the mainframe.
- Get started with z/OS Connect EE
  Learn how to install, configure, and get up and running with z/OS Connect Enterprise Edition.
- Test your APIs to z/OS assets
  Learn what questions to ask when testing APIs that expose z/OS assets. This includes thoughts on scalability, integration, test types, and available tools.
- Security
  Learn how to secure your APIs and API Requesters using a combination of Liberty for z/OS features (such as Security Access Facility), and z/OS Connect EE security capabilities.
- Managing API workloads
- DevOps with z/OS Connect EE
  Enterprises need a DevOps process to support agile development, testing, and deployment of services and APIs. When changes are made to your z/OS Connect EE services, APIs, or API requesters, you can use the z/OS Connect EE build toolkit, together with your source code management (SCM) system and DevOps solution, to support updates, testing, and...
- Open Banking

71

/questions?thanks=true

Thank you for listening.

72