IBM z/OS Connect Enterprise Edition

Security

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

IBM Z
Wildfire Team –
Washington System Center

© 2017, 2020 IBM Corporation

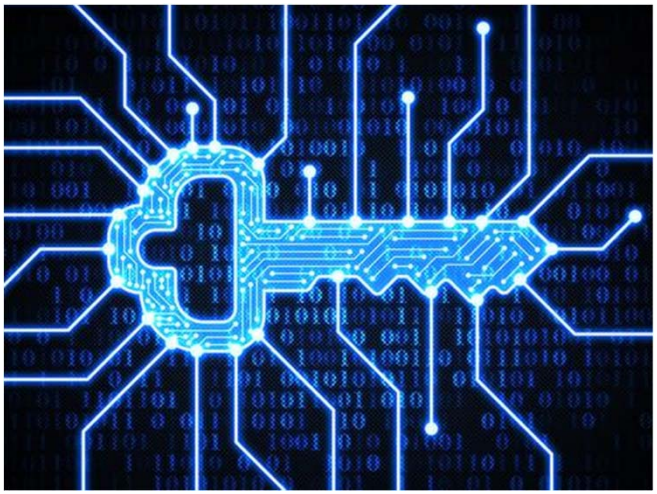
Contents

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

© 2017, 2020 IBM Corporation

General considerations for securing REST APIs

- 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**)



© 2017, 2020 IBM Corporation

Common challenges

- **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

© 2017, 2020 IBM Corporation

A review of Liberty security

© 2017, 2020 IBM Corporation

Review OMVS security - Unix file permissions

Owner

	Read	Write	Execute
Bit	1	1	1
Base-2 Value	[4]	[2]	[1]

4 + 2 + 1 =

7 The owner has READ, WRITE and EXECUTE

The **owner** of the file or directory

Group

	Read	Write	Execute
	1	0	1
	[4]	[2]	[1]

4 + 0 + 1 =

5 The group has READ and EXECUTE, but not WRITE

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

Other

	Read	Write	Execute
	0	0	0
	[4]	[2]	[1]

0 + 0 + 0 =

0 Others have nothing

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

© 2017, 2020 IBM Corporation

Default server configuration – server create zceesrv1

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

```
export JAVA_HOME=<path_to_64_bit_Java>
export WLP_USER_DIR=Servers1
./server create zceesrv1
```

```
/var/zosconnect          750  LIBSERV LIBGRP
└ /servers               750  LIBSERV LIBGRP
  ├ /.classcache         750  LIBSERV LIBGRP
  ├ /.logs               750  LIBSERV LIBGRP
  ├ /.pid                750  LIBSERV LIBGRP
  └ /zceesrv1            750  LIBSERV LIBGRP
    ├ /logs              750  LIBSERV LIBGRP
    │ └ messages.log     640  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:

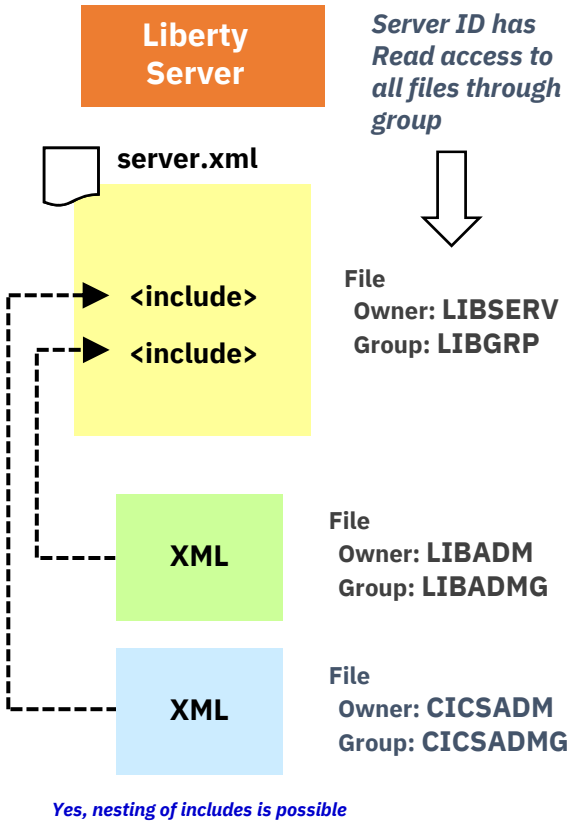
- If you have multiple people with a need to change configuration files, do you share the password of LIBSERV? (answer: **no**)
  Sharing passwords is a bad practice. Better to take advantage SAF SURROGAT so permitted users can switch to the owning ID so they can make changes
- If you have multiple people 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)

© 2017, 2020 IBM Corporation

Include file processing



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

© 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”/>
```

db2.xml (owned by a DBA)

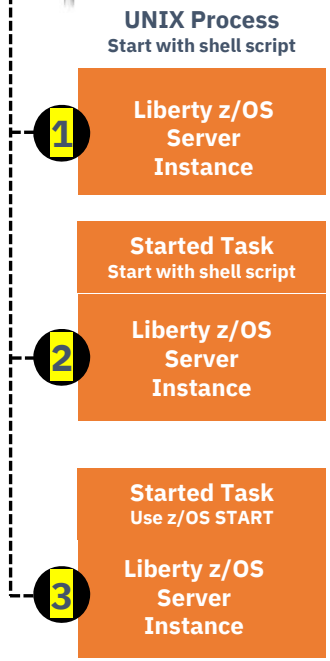
```
...
<server description="Db2 REST">

 <zosconnect_zosConnectServiceRestClientConnection
id="Db2Conn"
     host="wg31.washington.ibm.com"  port="2446"
    basicAuthRef="dsn2Auth" />

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

</server>
```


© 2017, 2020 IBM Corporation

z/OS : Starting Liberty Servers

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.



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

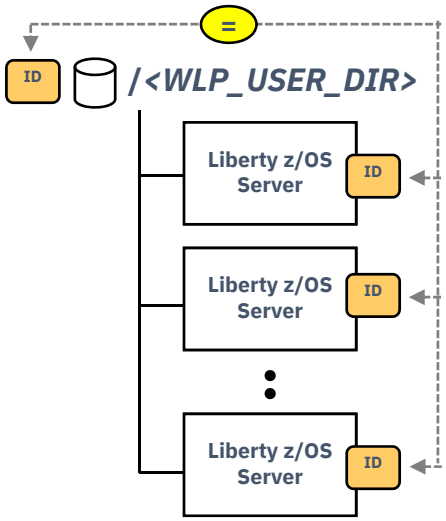
Expectation is for production servers either #2 (via Collective Controller) or #3 will be used

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

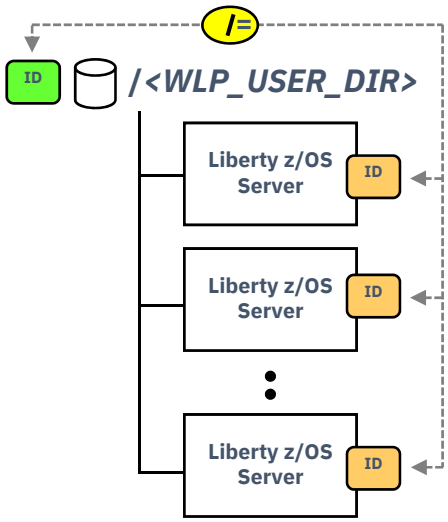
© 2017, 2020 IBM Corporation

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

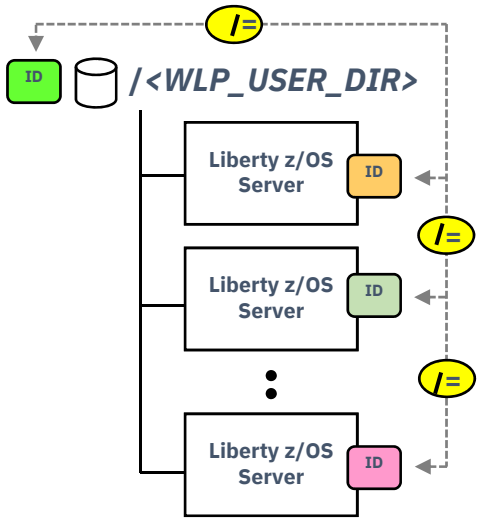
On z/OS, the best practice for Liberty servers in production is that they run as 'Started Tasks' (STCs).



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



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



- Multiple servers
- Different STC IDs
- STC IDs ≠ 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

© 2017, 2020 IBM Corporation

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

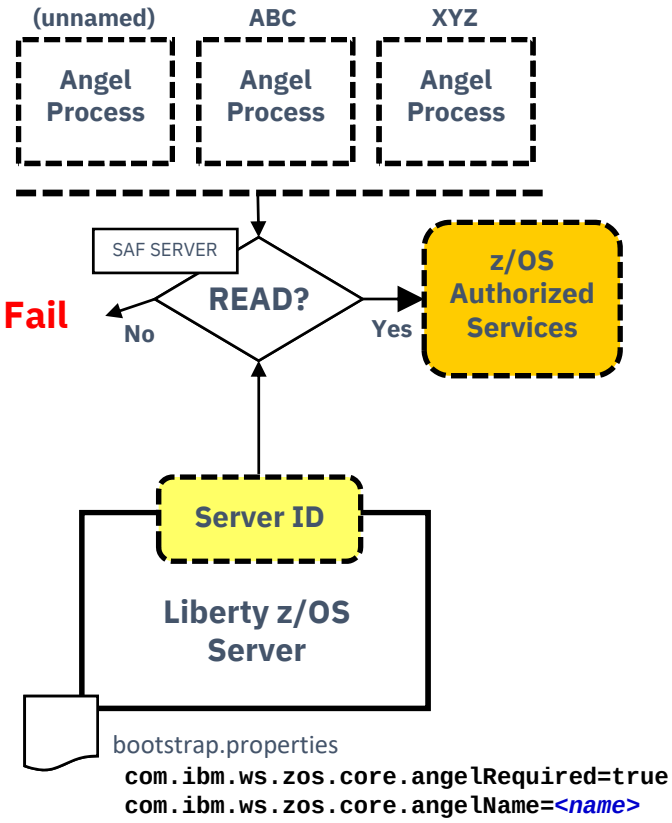
Then the second question is whether servers under a WLP_USER_DIR will share a common JCL start proc, or use unique start procs for each server

	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.

© 2017, 2020 IBM Corporation

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



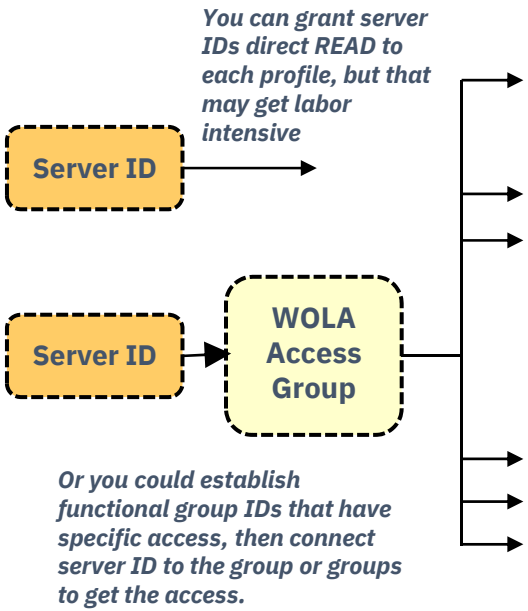
The Angel Process is a started task that is used to protect access to z/OS authorized services. This is done with SAF SERVER profiles.

- 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**

© 2017, 2020 IBM Corporation

z/OS : SAF SERVER profiles related to the Angel



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

BBG.AUTHMOD.BBGZSAFM.WOLA

BBG.AUTHMOD.BBGZSCFM.WOLA

enables use of WOLA

BBG.AUTHMOD.BBGZSAFM.PRODMGR enables use of IFAUSAGE services

BBG.AUTHMOD.BBGZSAFM.ZOSAIO enables use TCP asynchronous I/O services

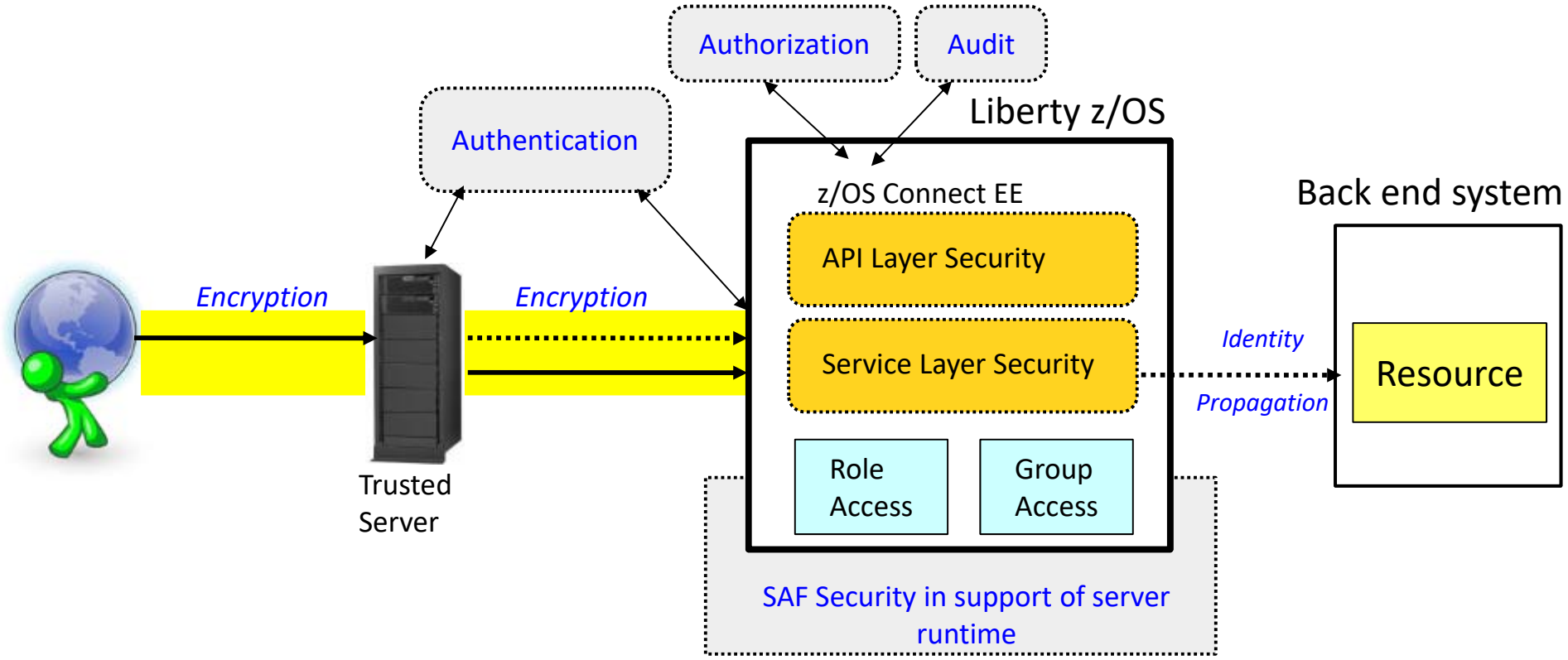
Best practice:

- **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**

© 2017, 2020 IBM Corporation

z/OS Connect EE API provider security overview

z/OS Connect EE

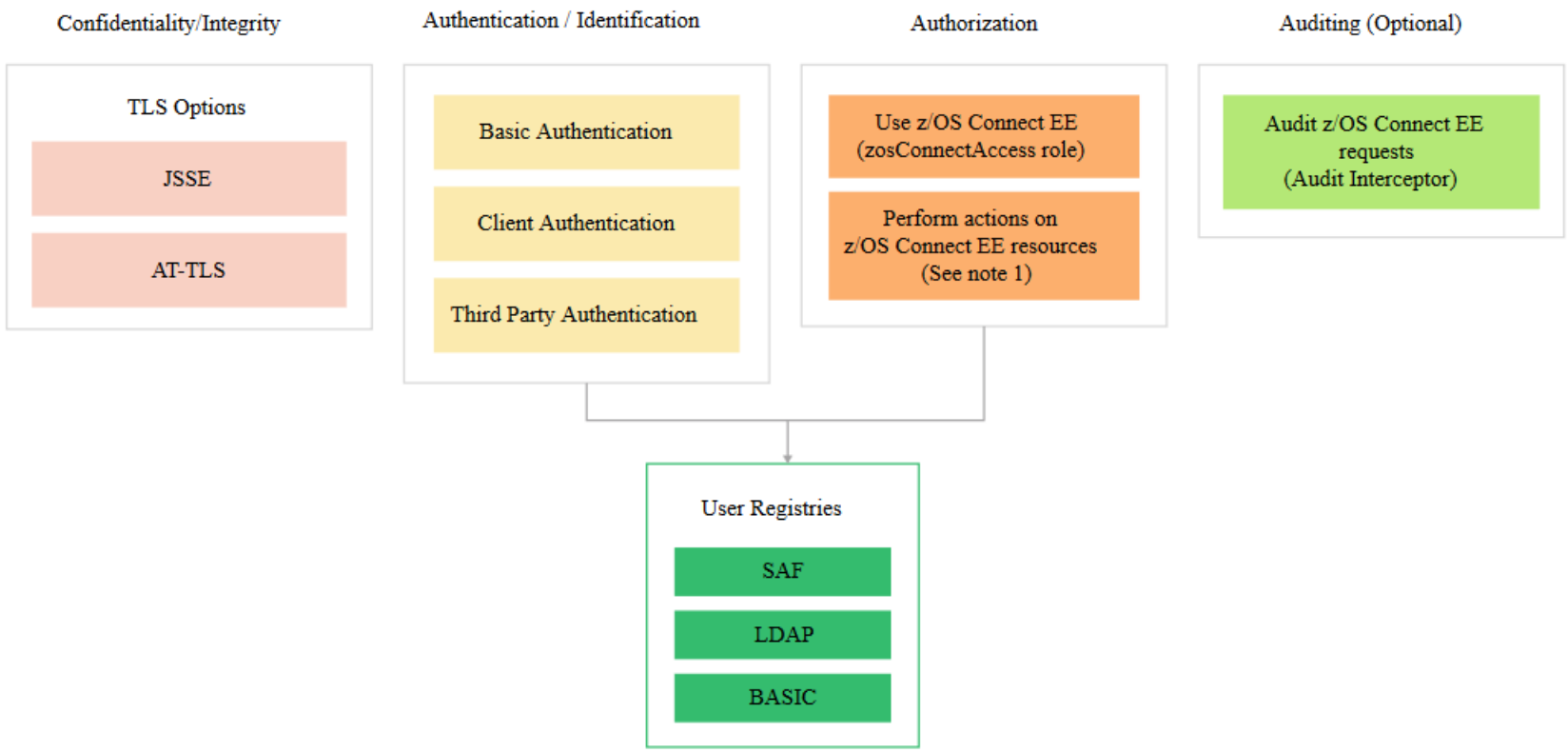


1. Authentication (basic, client certificates, 3rd party authentication)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (role and group access)
4. Audit
5. Configuring security with SAF
6. Back end identity propagation (CICS, IMS, Db2, MQ)

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

© 2017, 2020 IBM Corporation

z/OS Connect EE security options



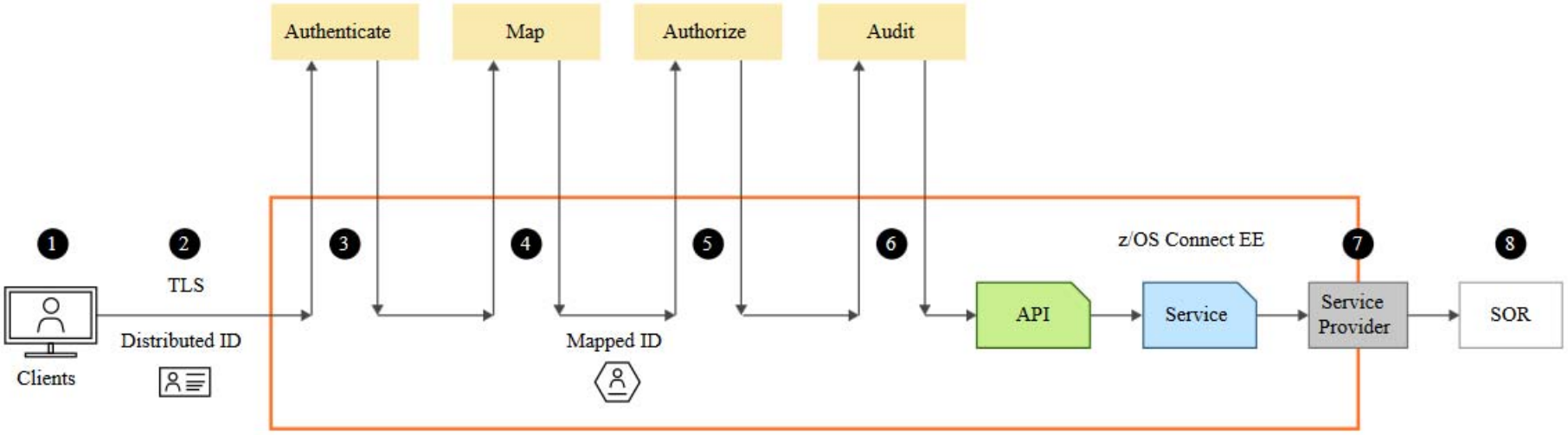
http://ibm.biz/zosconnect-security

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.

© 2017, 2020 IBM Corporation

Typical z/OS Connect EE security flow

z/OS Connect EE



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

© 2017, 2020 IBM Corporation

Authentication

Obtaining an identity

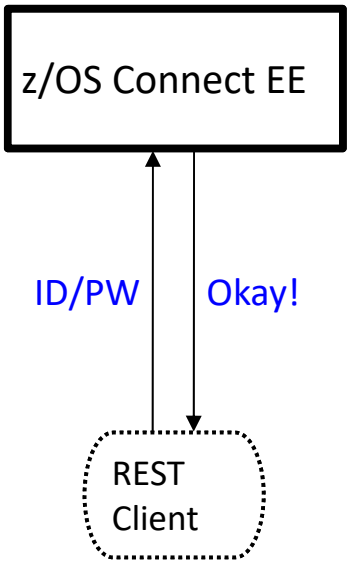
© 2017, 2020 IBM Corporation

Authentication

z/OS Connect EE

Several different ways this can be accomplished:

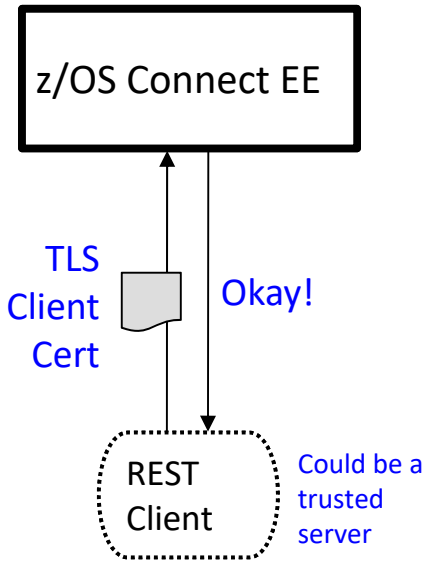
Basic Authentication



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

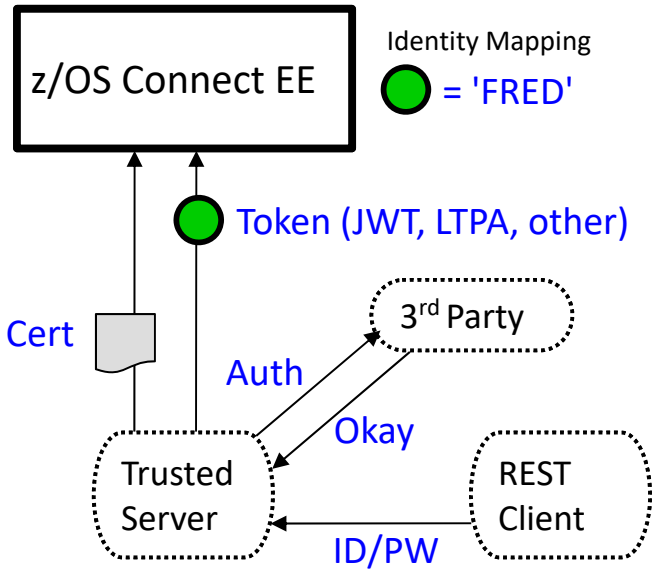
- Basic (server.xml)
- LDAP
- SAF

Client Certificate



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

Third Party Authentication



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

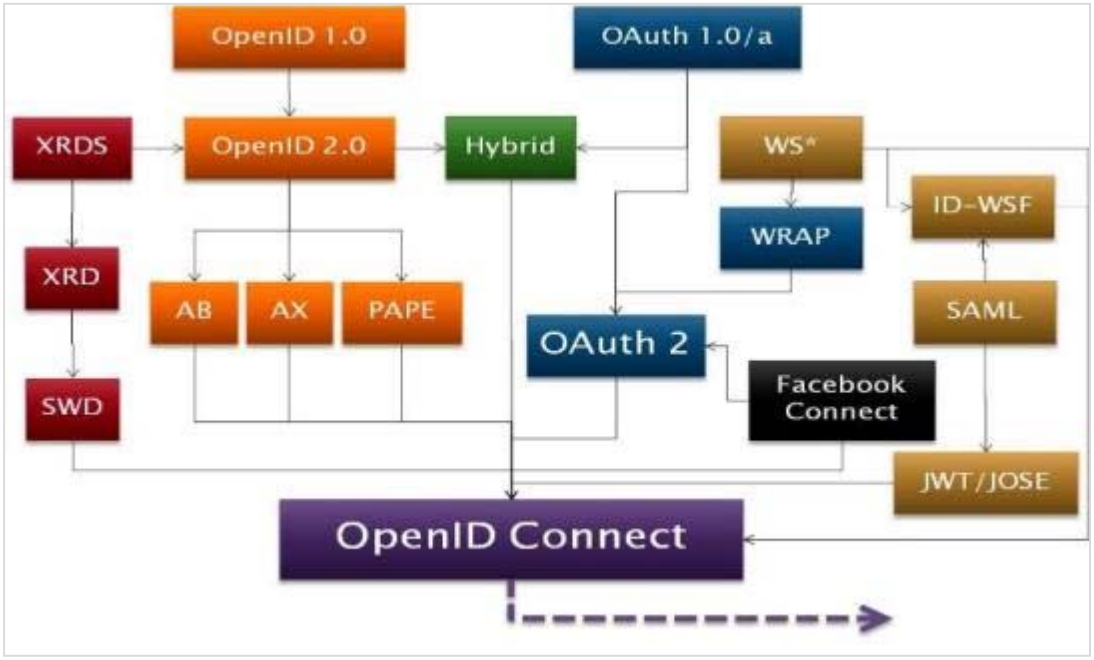
© 2017, 2020 IBM Corporation

Security token types by 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	

© 2017, 2020 IBM Corporation

Open security standards



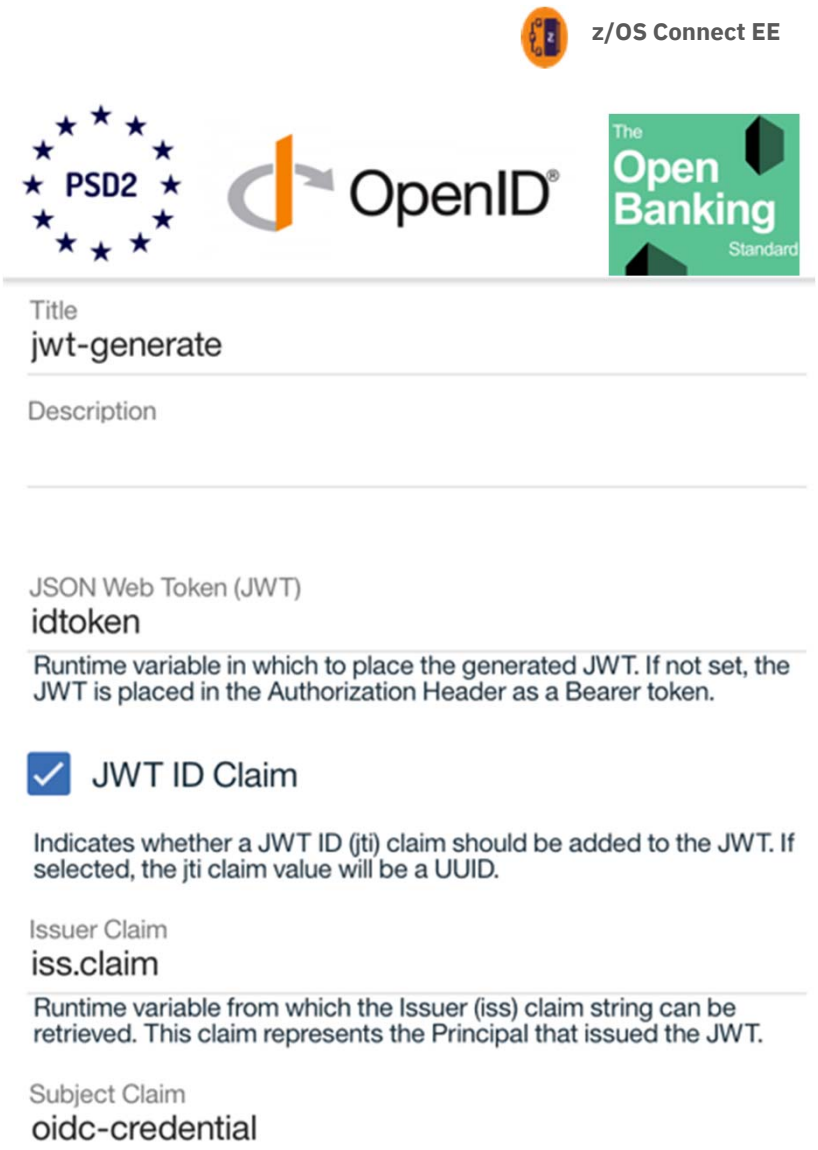
- **OAuth** is an open standard for secure delegated access to server resources designed to work with HTTP
- **OpenID Connect** is an authentication layer on top of Oauth
- **JWT** (JSON Web token) defines a compact and self-contained way for securely transmitting information between parties as a JSON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

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

© 2017, 2020 IBM Corporation

OpenID Connect Overview

- **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



© 2017, 2020 IBM Corporation

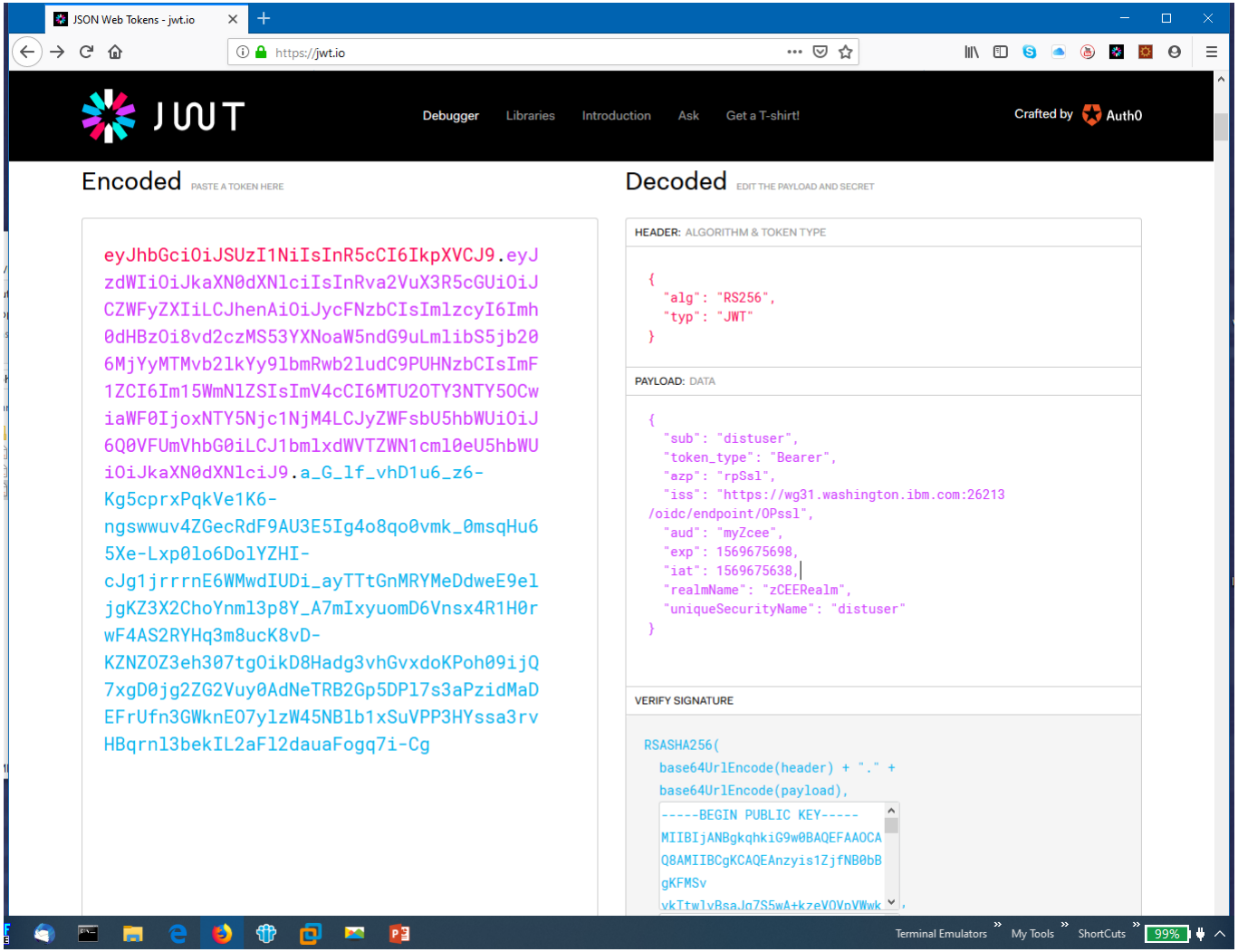
Why JWT with z/OS Connect EE?

- 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

© 2017, 2020 IBM Corporation

JWT (JSON Web Token)

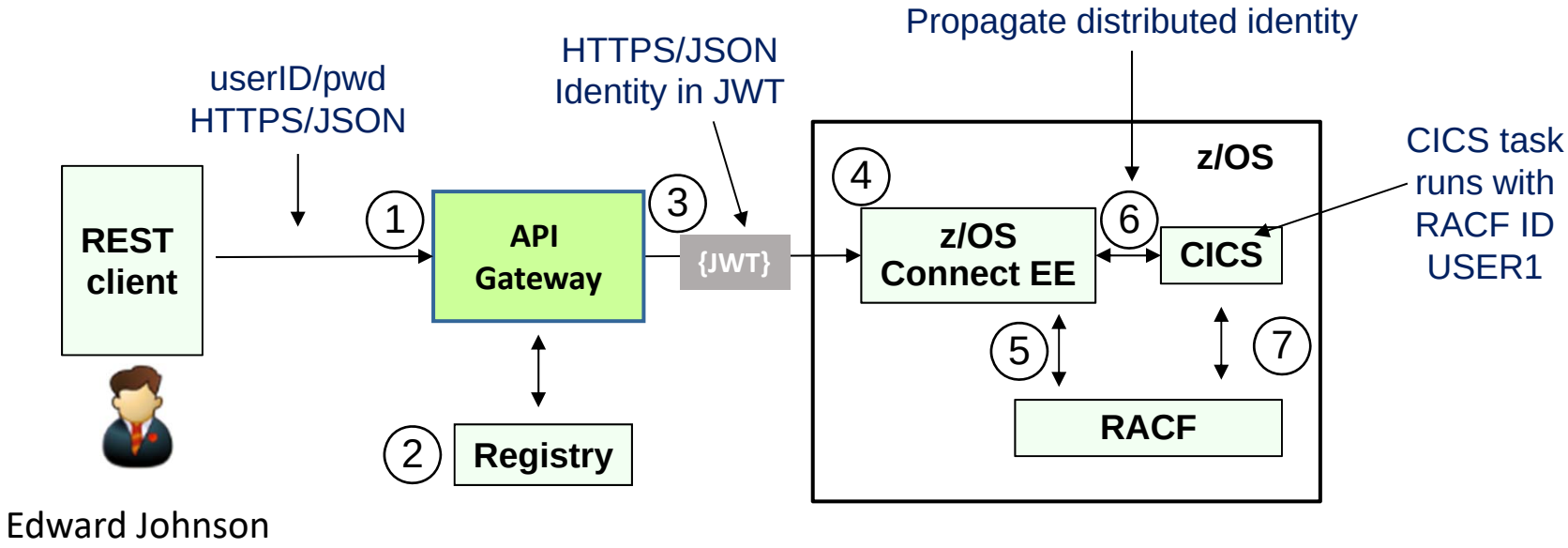
- JWT is a compact way of representing claims that are to be transferred between two parties
- Normally transmitted via HTTP header
- Consists of three parts
  - Header
  - Payload
  - Signature



© 2017, 2020 IBM Corporation

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

© 2017, 2020 IBM Corporation

JWT used in scenario


```
{
 "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

© 2017, 2020 IBM Corporation

Configuring authentication with JWT

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

© 2017, 2020 IBM Corporation

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.

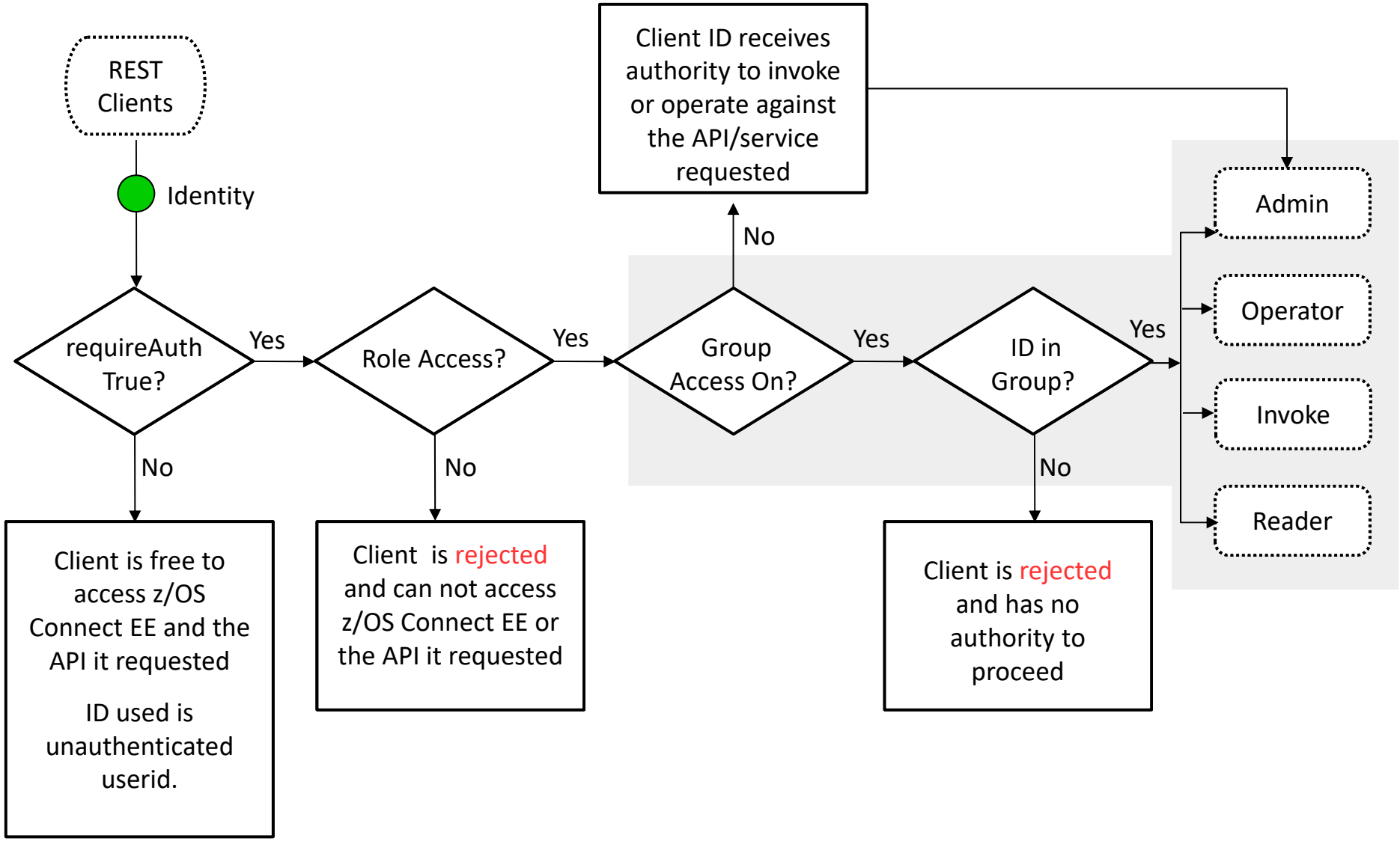
© 2017, 2020 IBM Corporation

Authorization

Once we have an identity

© 2017, 2020 IBM Corporation

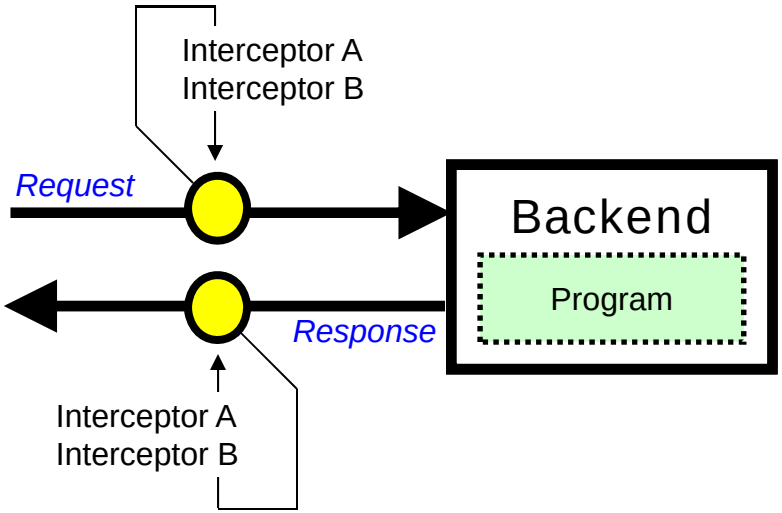
Security flow with z/OS Connect EE



© 2017, 2020 IBM Corporation

Overview of z/OS Connect interceptors

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



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)

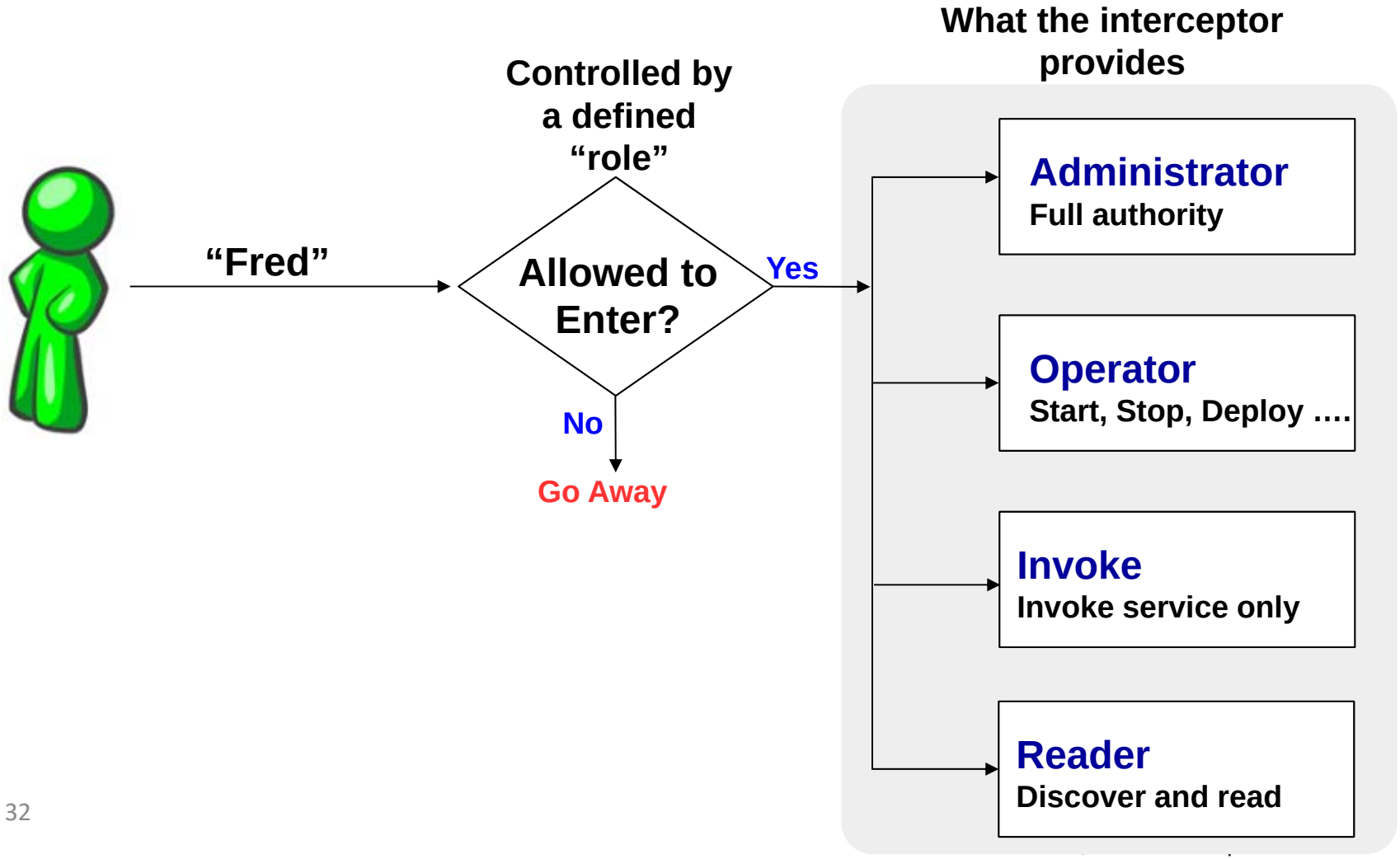
It is also possible to write your own interceptor and have it called as part of request/response processing

© 2017, 2019 IBM Corporation

Authorization interceptor

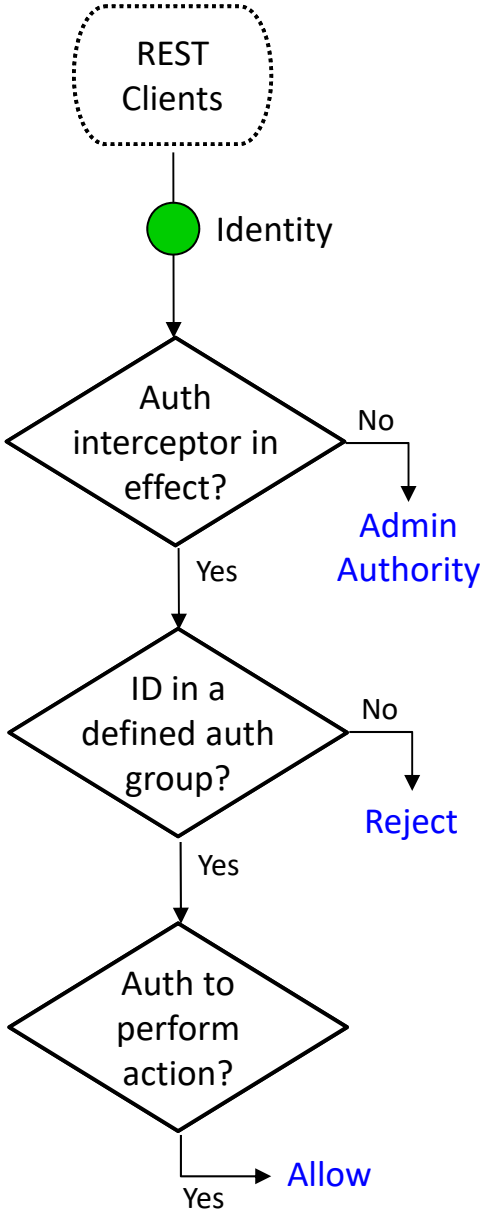
z/OS Connect EE

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:



Configuring the authorization interceptor

z/OS Connect EE



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.

© 2017, 2020 IBM Corporation

Audit

© 2017, 2019 IBM Corporation

Audit (SMF) Interceptor

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

© 2017, 2020 IBM Corporation

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.

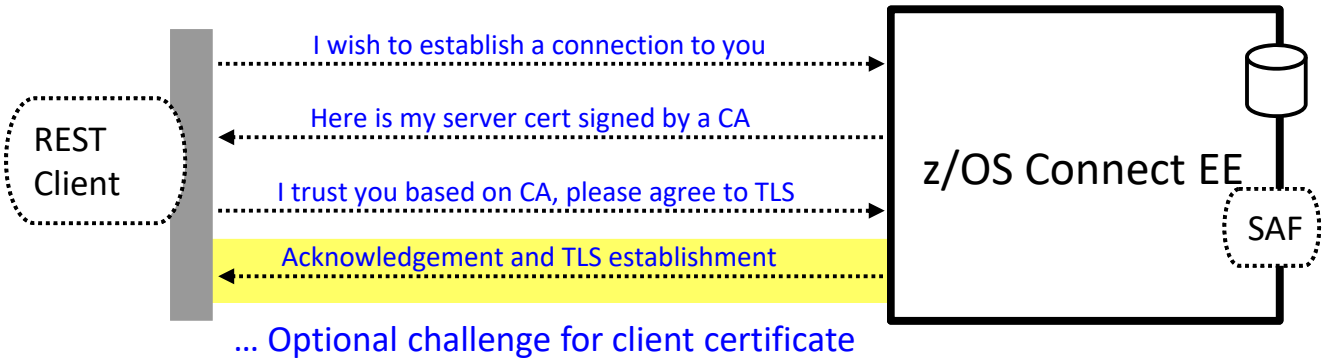
```
<zosconnect_zosConnectManager globalInterceptorsRef="interceptorList_g"
globalAdminGroup="GMADMIN" globalInvokeGroup="GMINVOKE"/>

<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>
```


© 2017, 2020 IBM Corporation

Encryption

© 2017, 2020 IBM Corporation

SSL/TLS connections



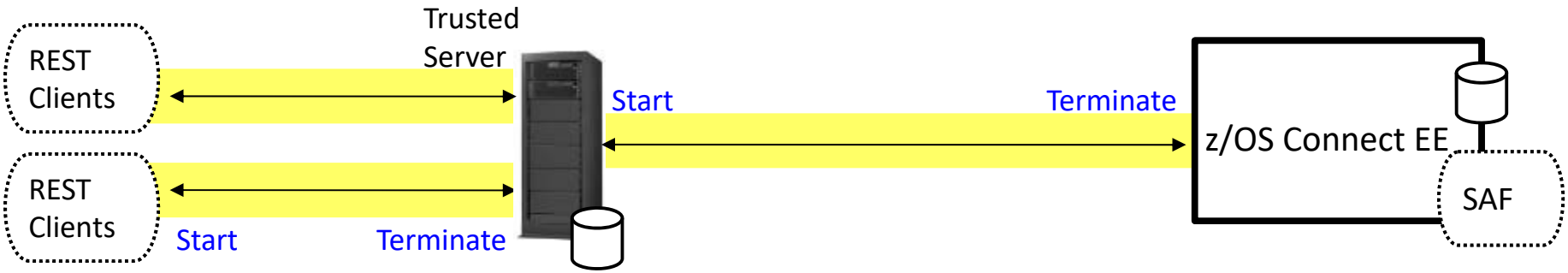
Java-based key/trust files

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:



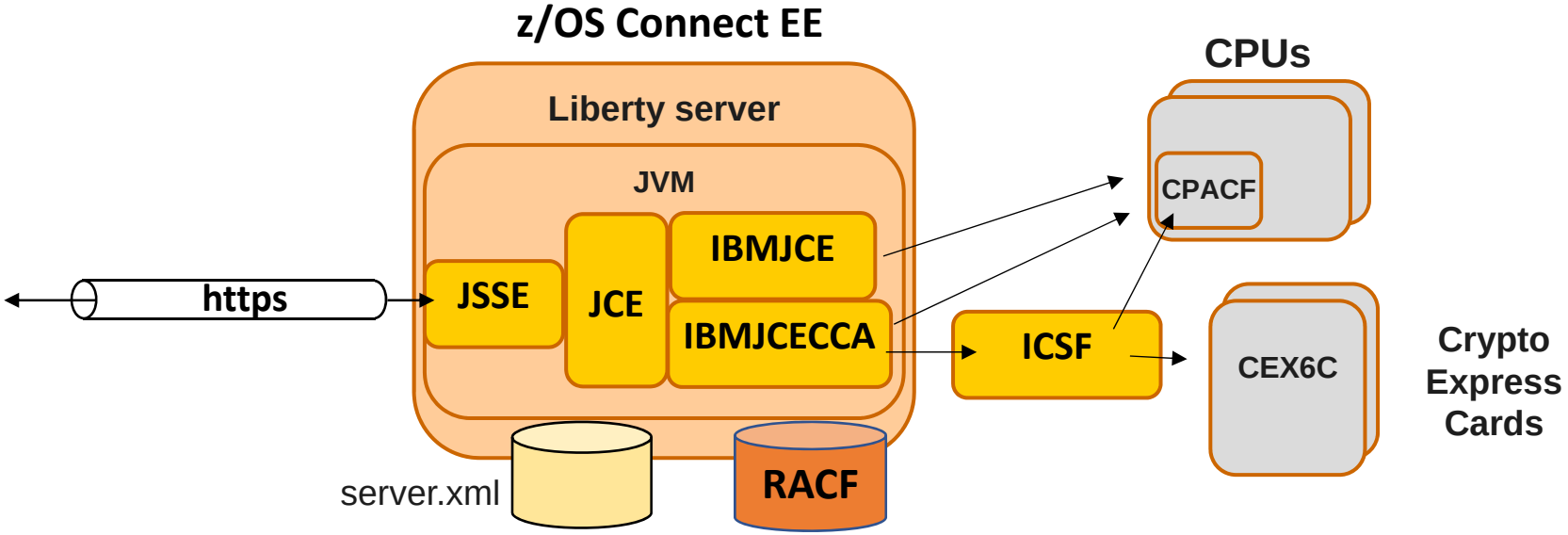
The client TLS sessions may come and go frequently. If that's the responsibility of a mid-tier trusted server, then the overhead of setup/teardown is there, not on the z/OS system

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

© 2017, 2020 IBM Corporation

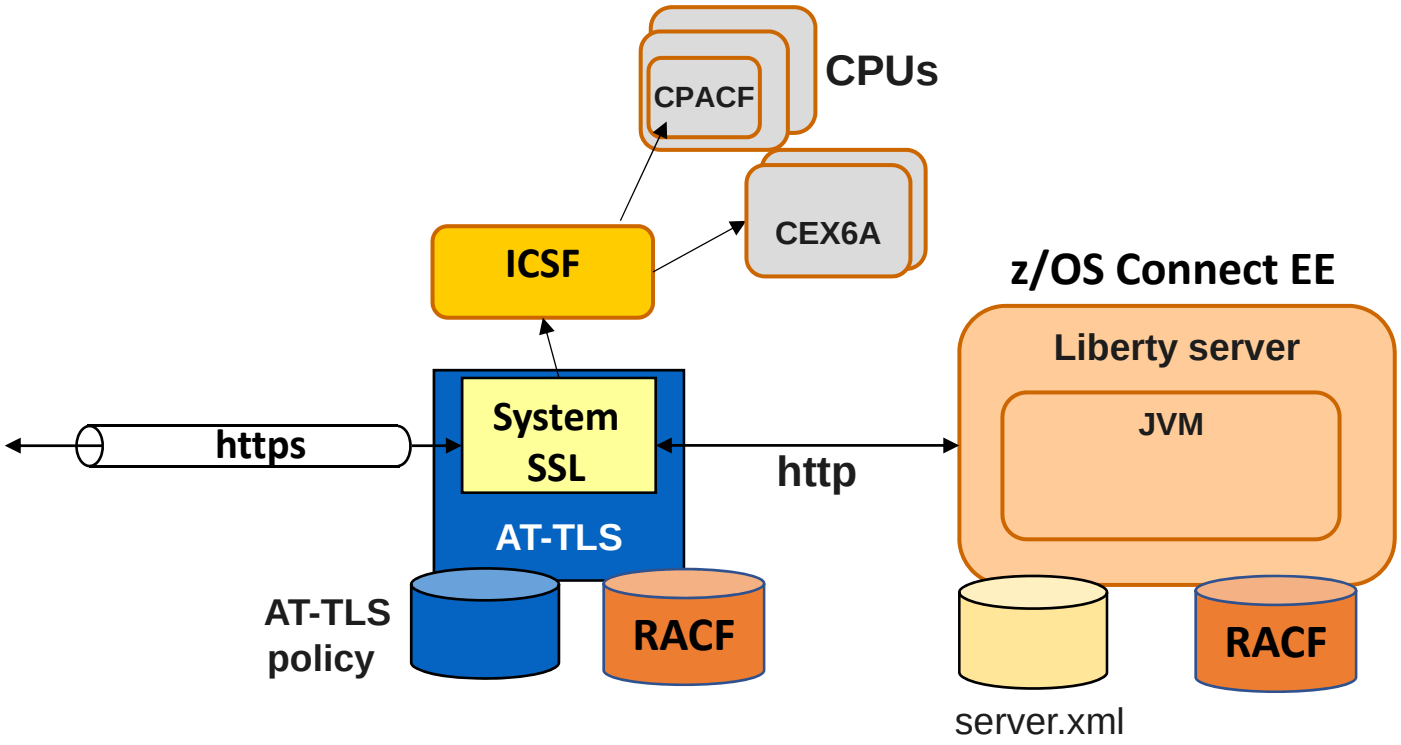
Using JSSE with z/OS Connect EE



- 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**

© 2017, 2020 IBM Corporation

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

© 2017, 2020 IBM Corporation

JSSE and AT-TLS comparison

z/OS Connect EE

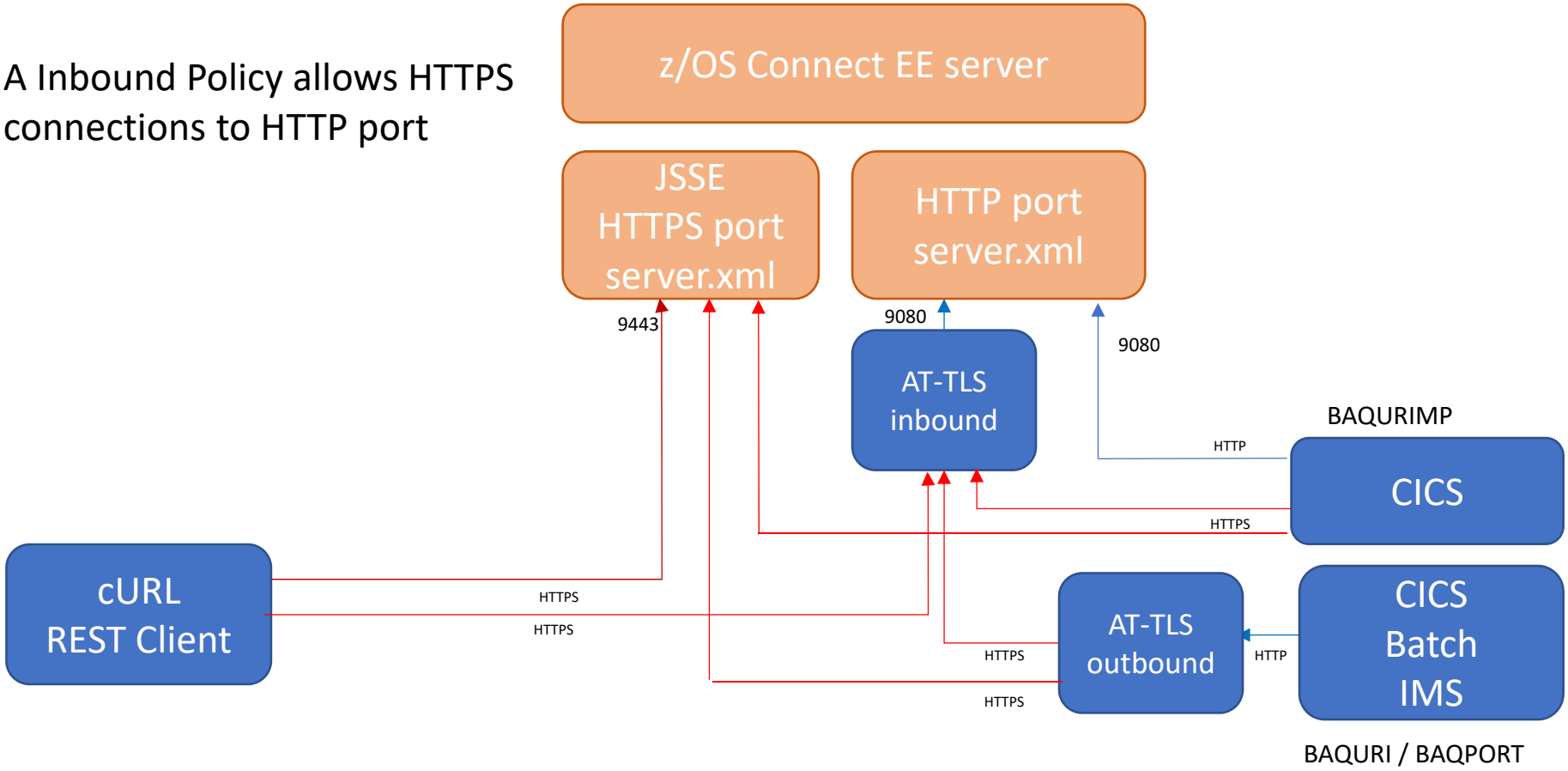
Capability	Description	JSSE	AT-TLS
1-way SSL	Verification of z/OS Connect certificate by client	Yes	Yes
2-way SSL	Verification of client certificate by z/OS Connect	Yes	Yes
SSL client authentication	Use of client certificate for authentication	Yes	**No**
Support for requireSecure option on APIs	Requires 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

© 2017, 2020 IBM Corporation

AT-TLS Inbound to zCEE Scenarios

z/OS Connect EE

A Inbound Policy allows HTTPS connections to HTTP port

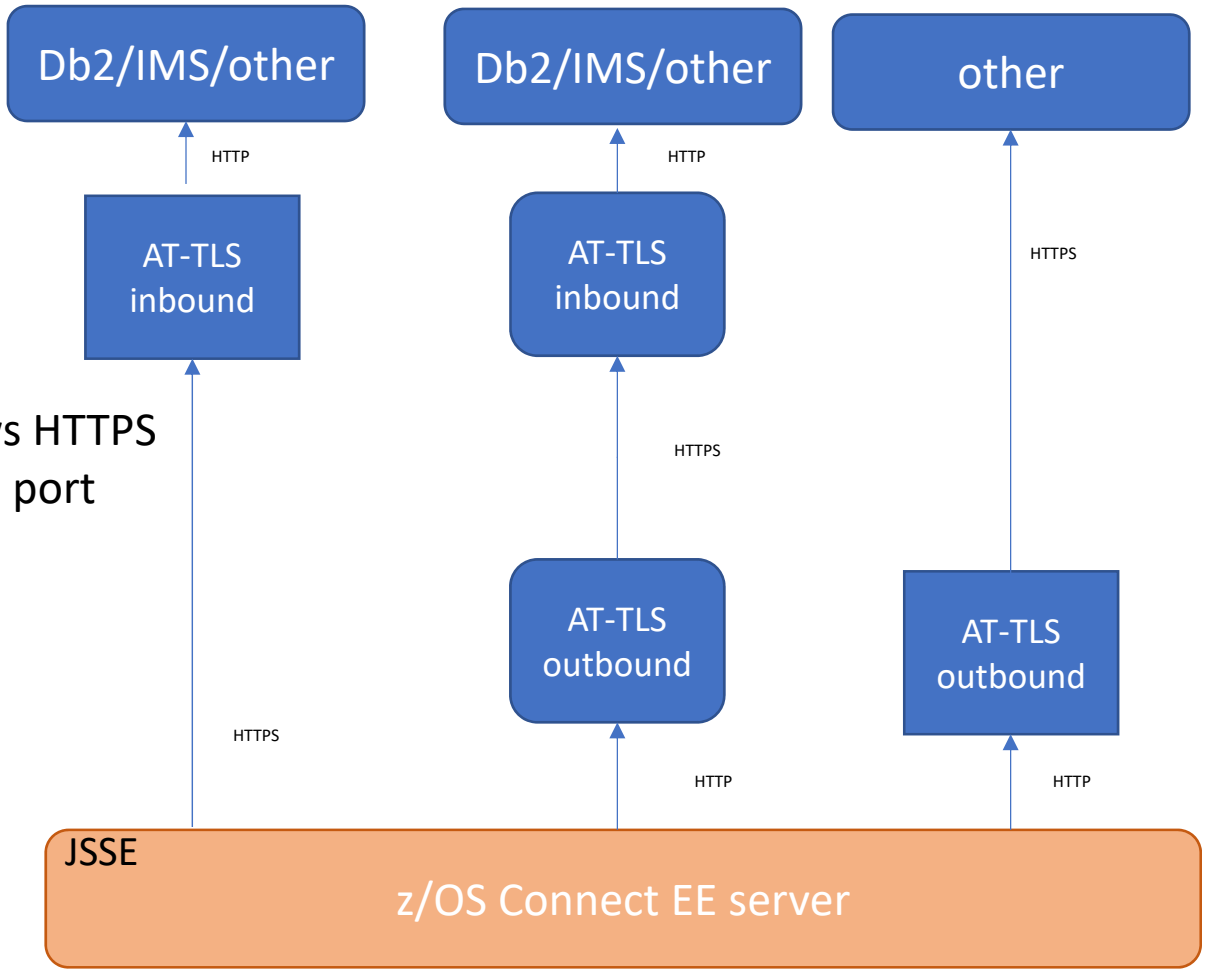


A Outbound Policy allows non-SSL clients to connection to HTTPS ports

© 2017, 2020 IBM Corporation

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

z/OS Connect EE



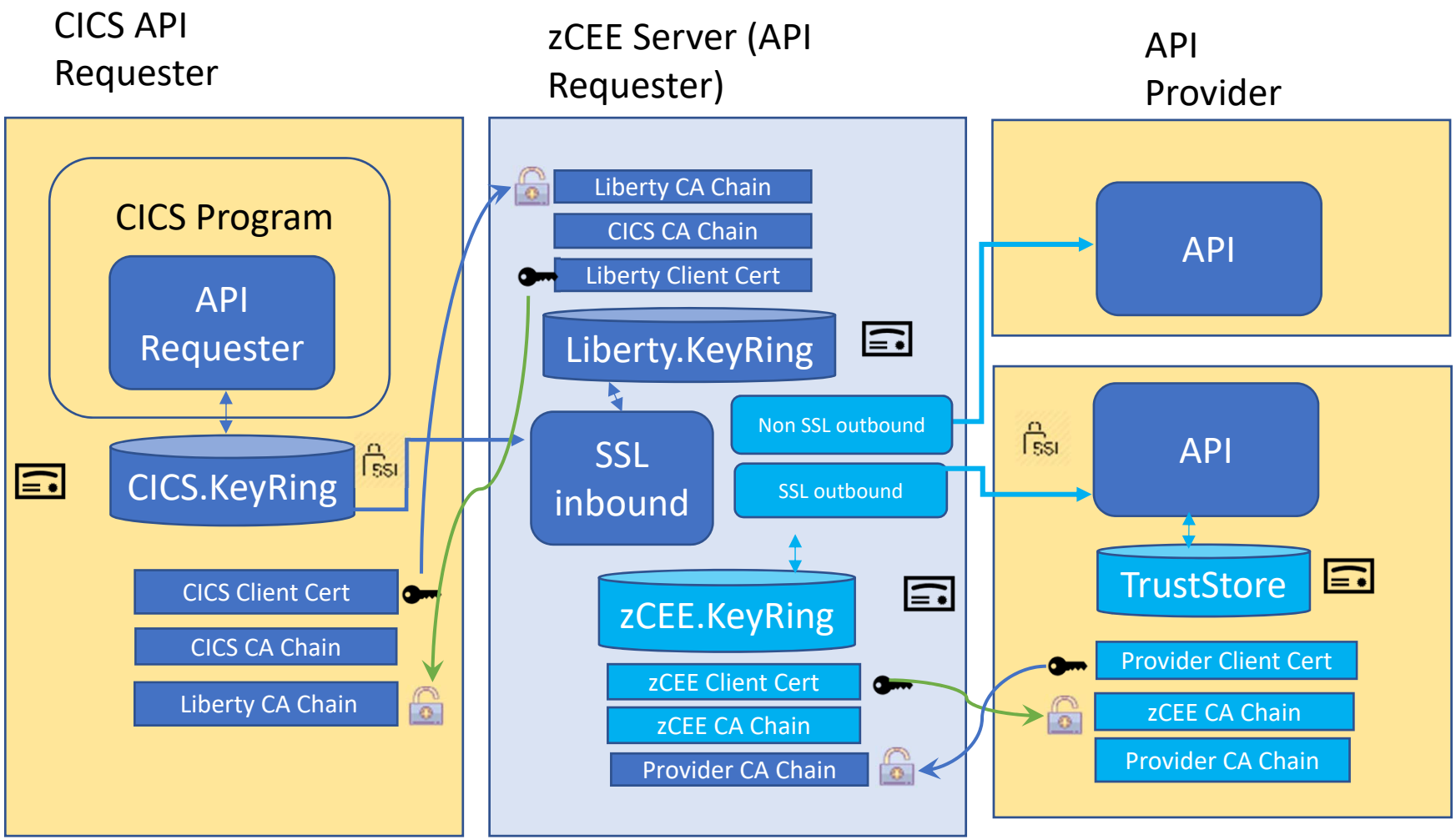
Inbound Policy allows HTTPS connections to HTTP port

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

© 2017, 2020 IBM Corporation

TLS Handshake Flow

z/OS Connect EE



© 2017, 2020 IBM Corporation

Configuring TLS Encryption with JSSE

© 2017, 2020 IBM Corporation

Cyphers

- 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

© 2017, 2020 IBM Corporation

Persistent connections

- 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

© 2017, 2020 IBM Corporation

SSL sessions

- 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

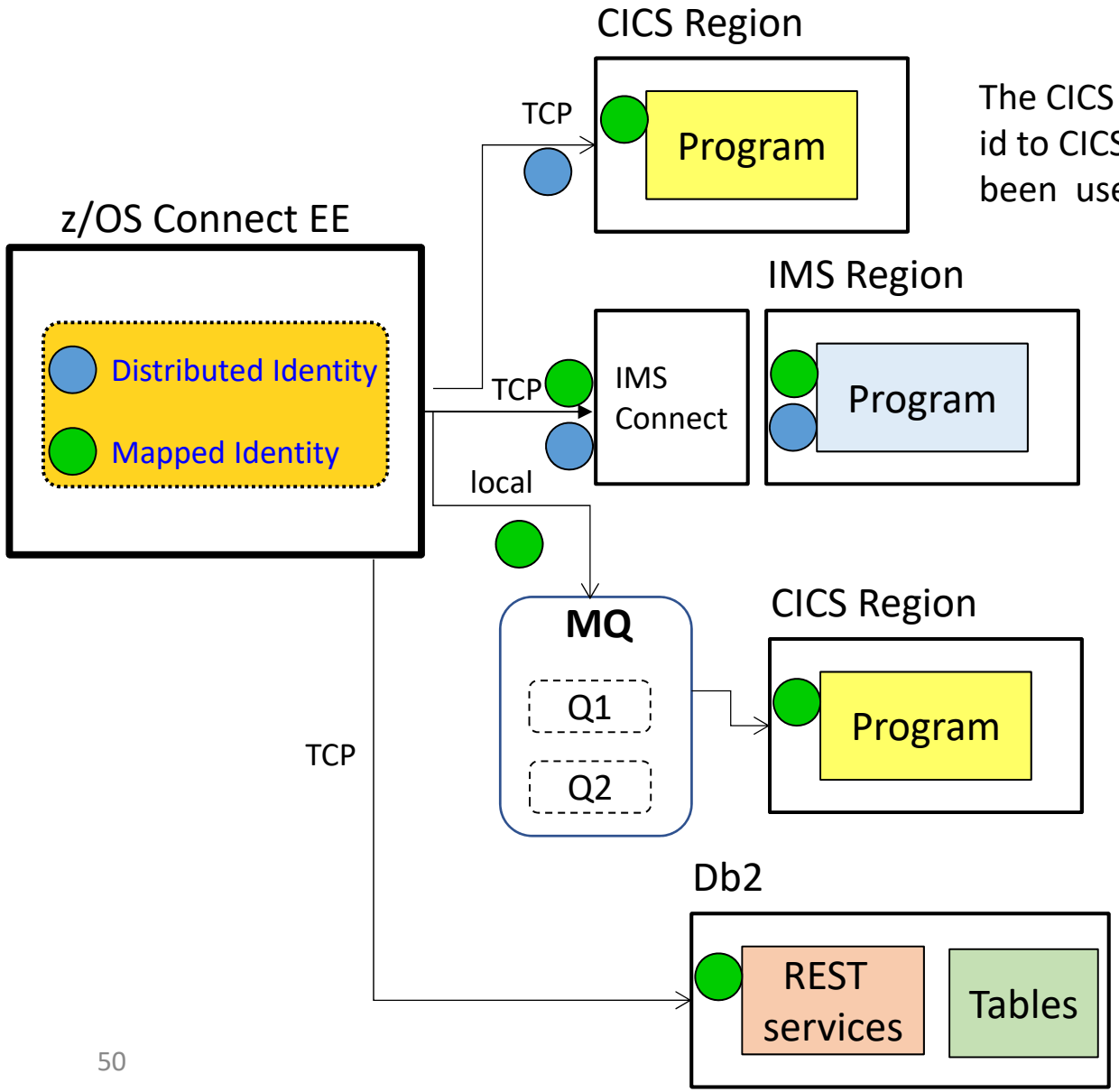
© 2017, 2020 IBM Corporation

Flowing identities to back end systems

© 2017, 2020 IBM Corporation

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

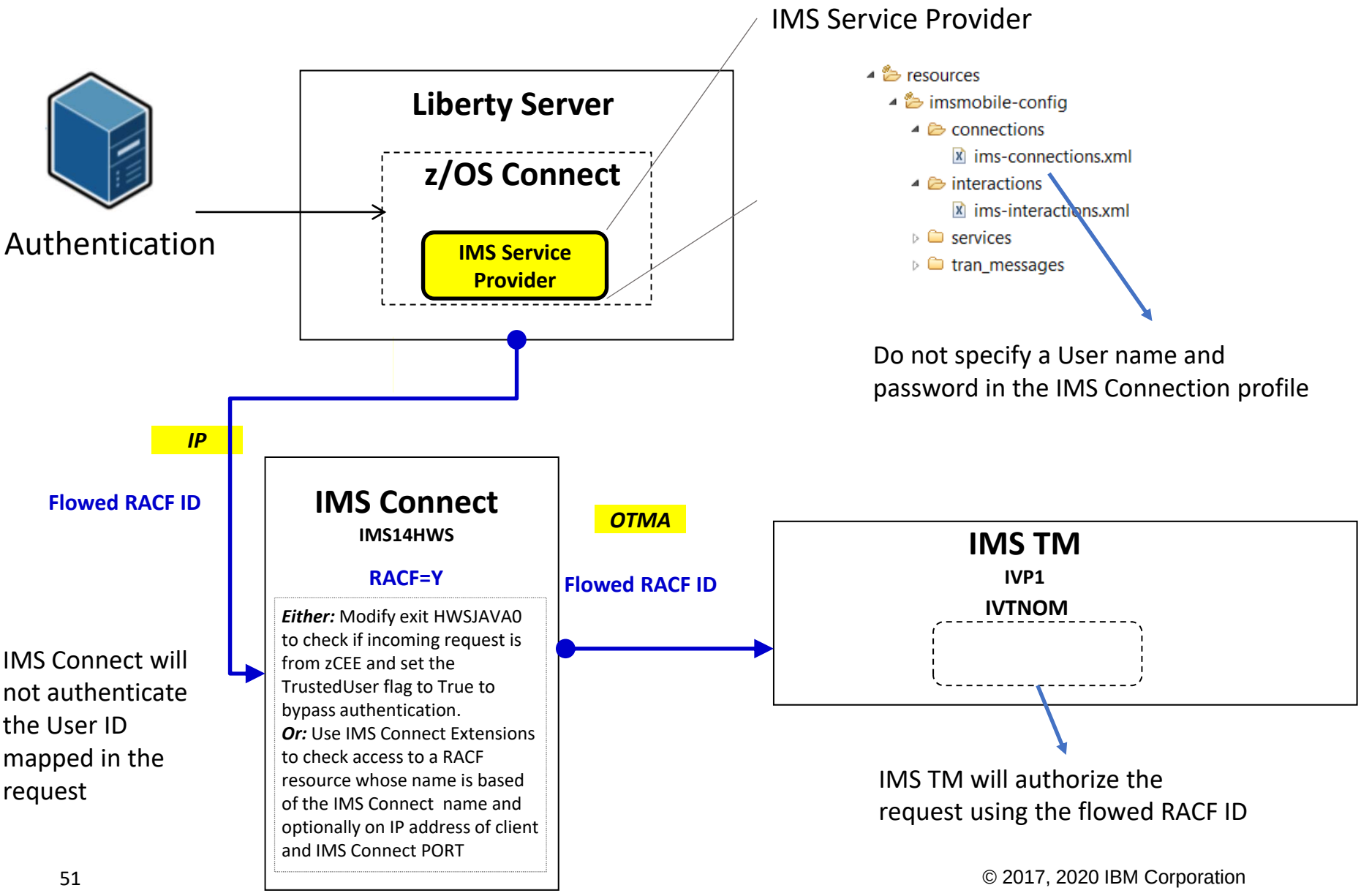
The MQ SP asserts the mapped id to CICS

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

© 2017, 2019 IBM Corporation

Flowing a RACF ID to IMS

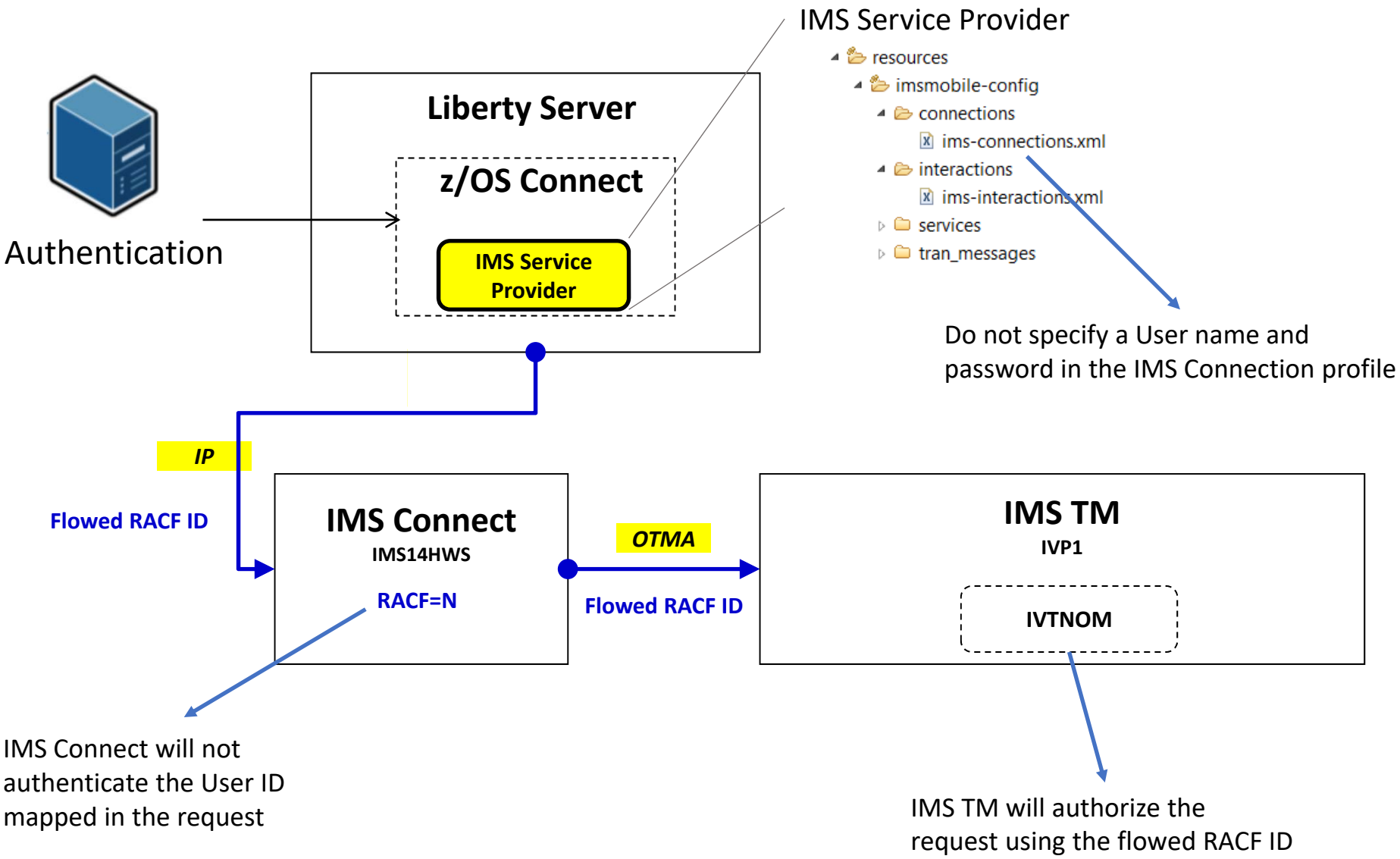
z/OS Connect EE



© 2017, 2020 IBM Corporation

Flowing a RACF ID to IMS

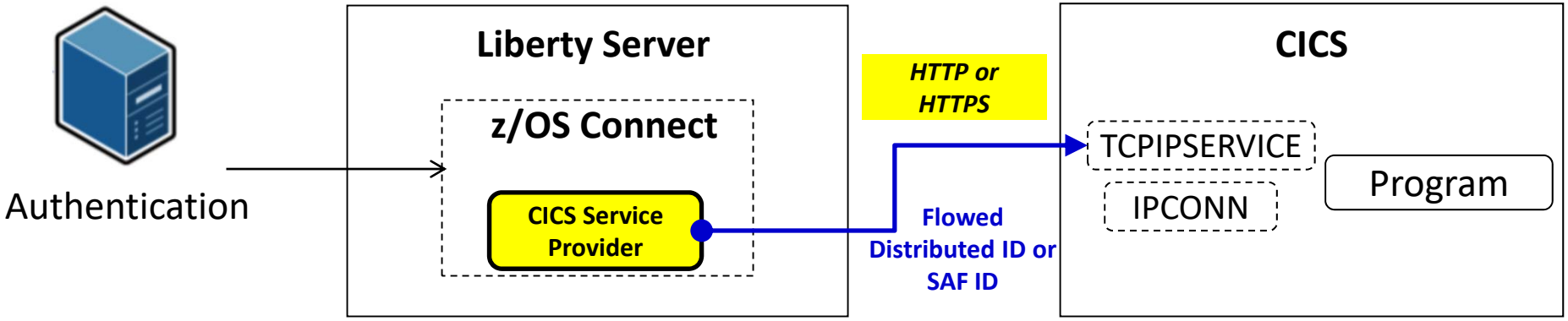
z/OS Connect EE



© 2017, 2020 IBM Corporation

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.

© 2017, 2020 IBM Corporation

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

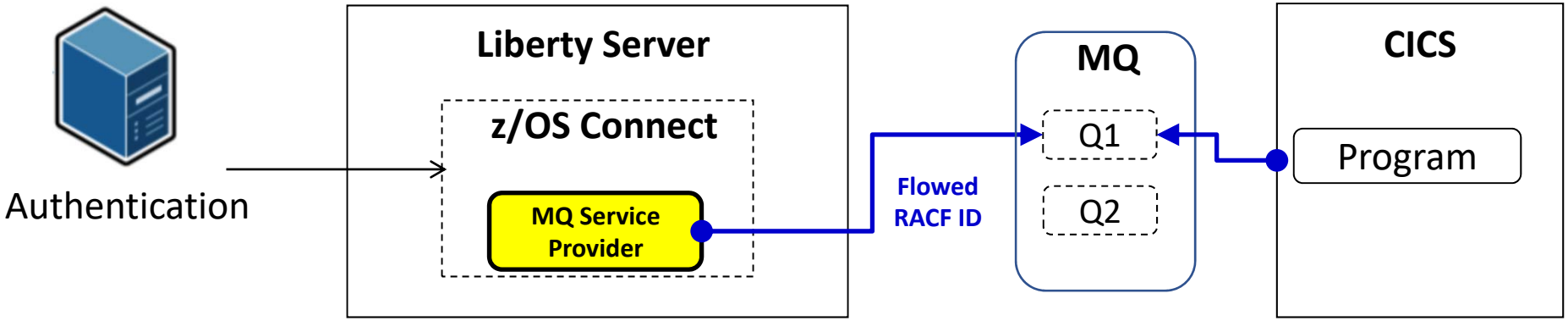
Requests run under the flowed user ID

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


© 2017, 2020 IBM Corporation

Flowing a user ID with MQ service provider

z/OS Connect EE



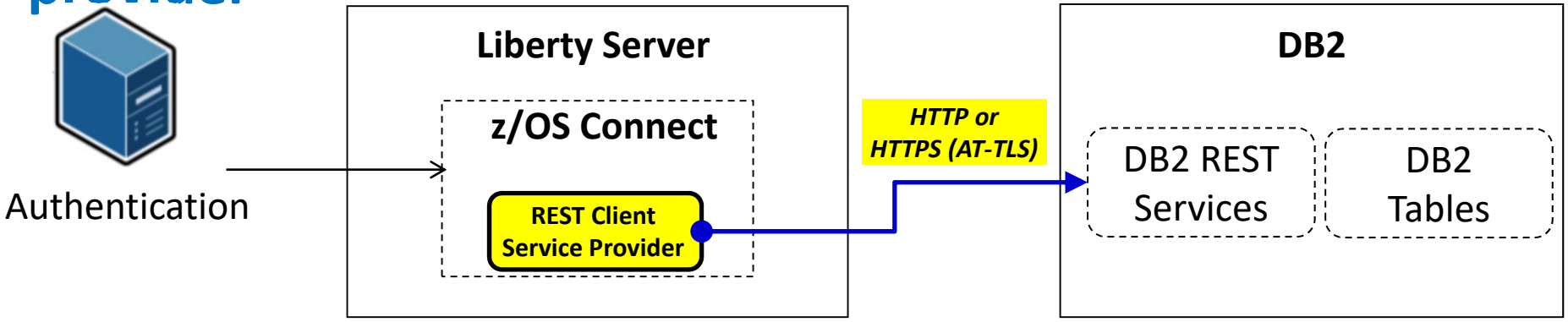
Set **`useCallerPrincipal=true`** to flow the authenticated RACF user ID

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


© 2017, 2020 IBM Corporation

Setting the user ID for the REST client service provider

z/OS Connect EE




```
<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

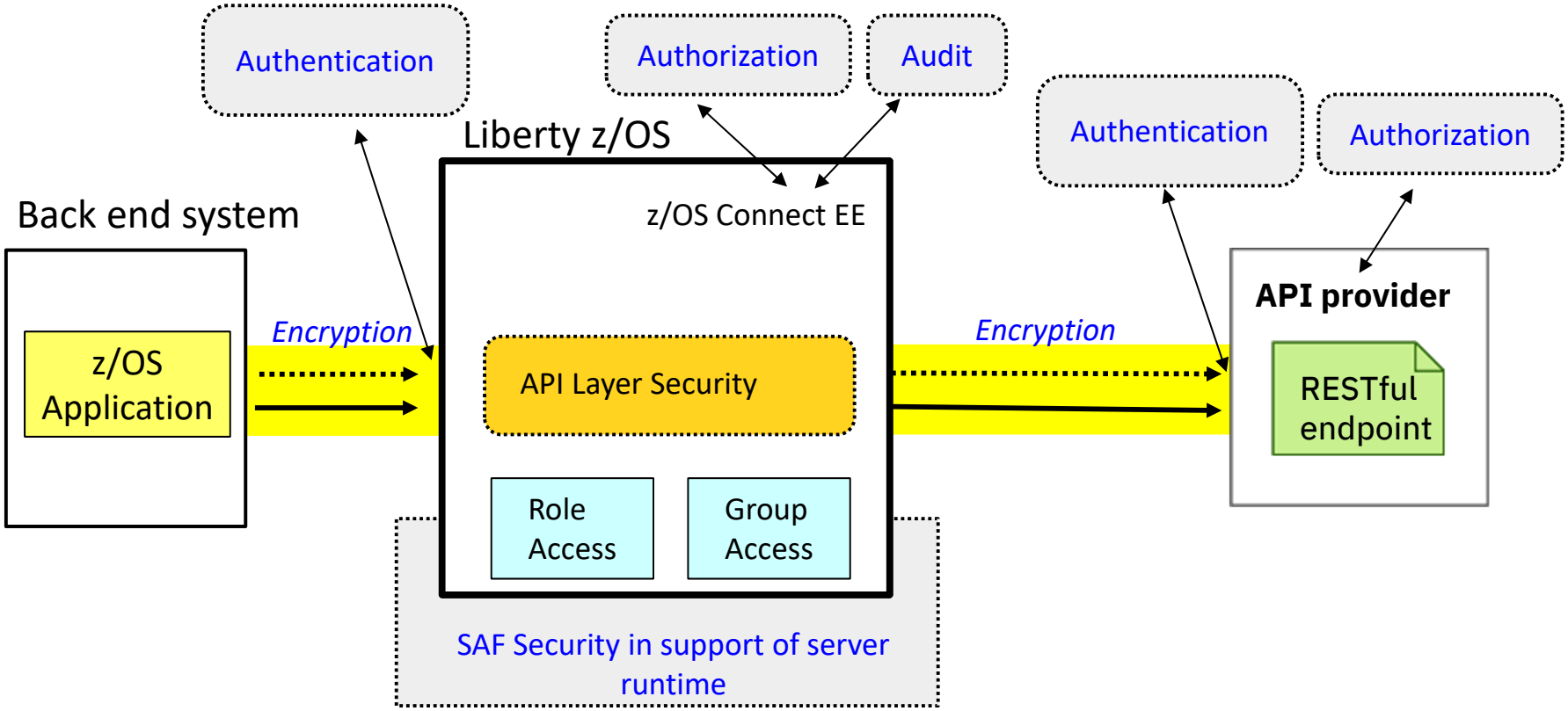
© 2017, 2020 IBM Corporation

What's different for API Requester?

© 2017, 2020 IBM Corporation

API requester security – overview

z/OS Connect EE

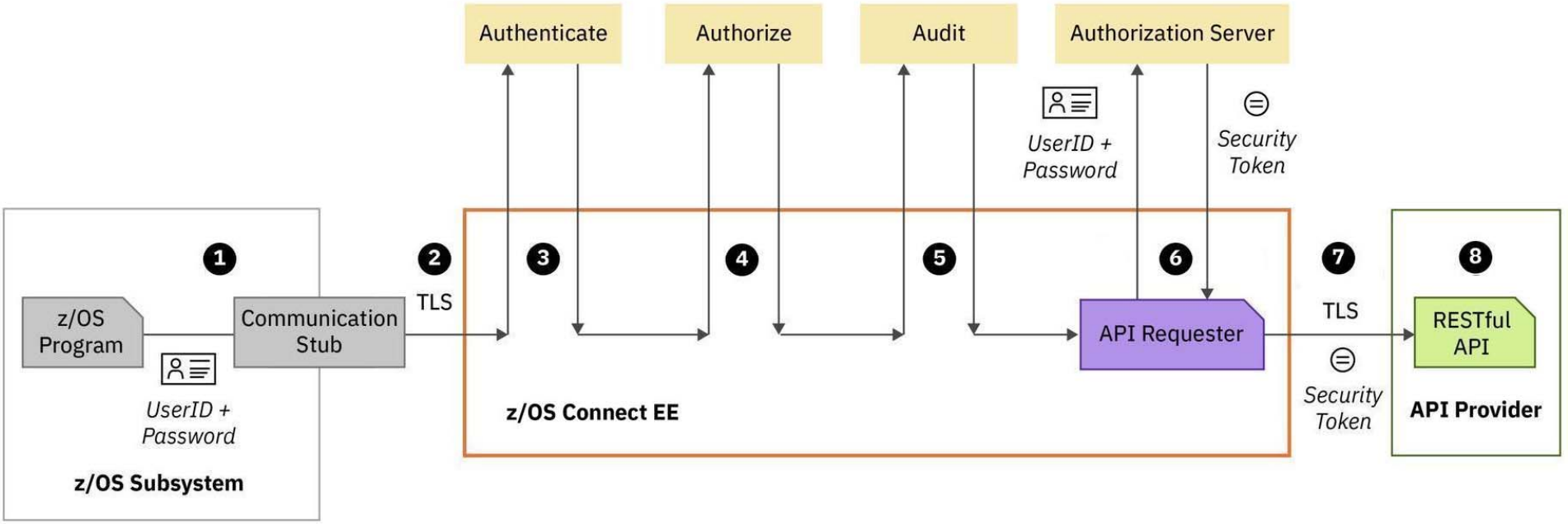


1. Authentication (basic, client certificate)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (OAuth)
4. Audit
5. Configuring security with SAF

© 2017, 2020 IBM Corporation

Typical z/OS Connect EE security flow

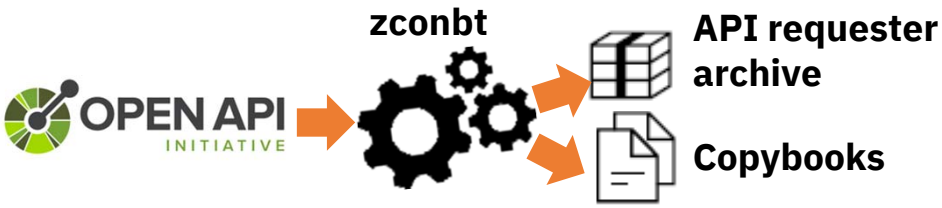
z/OS Connect EE

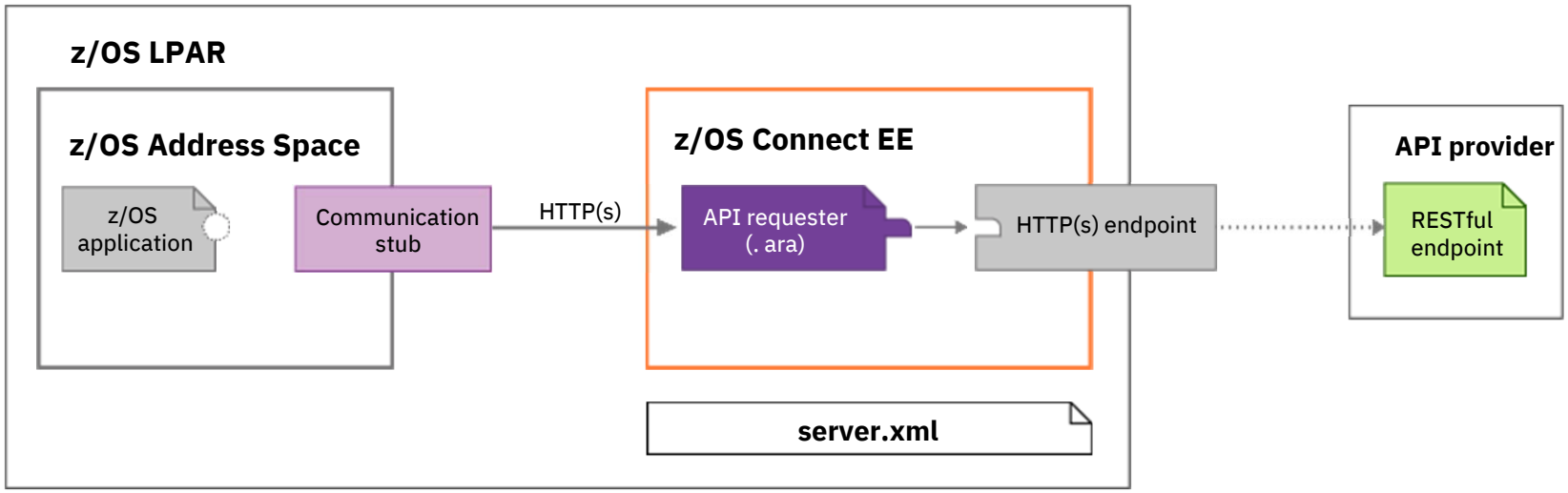


1. A user ID and password can be used for basic authentication by the z/OS Connect EE server
2. Connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to z/OS 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

© 2017, 2020 IBM Corporation

Authentication





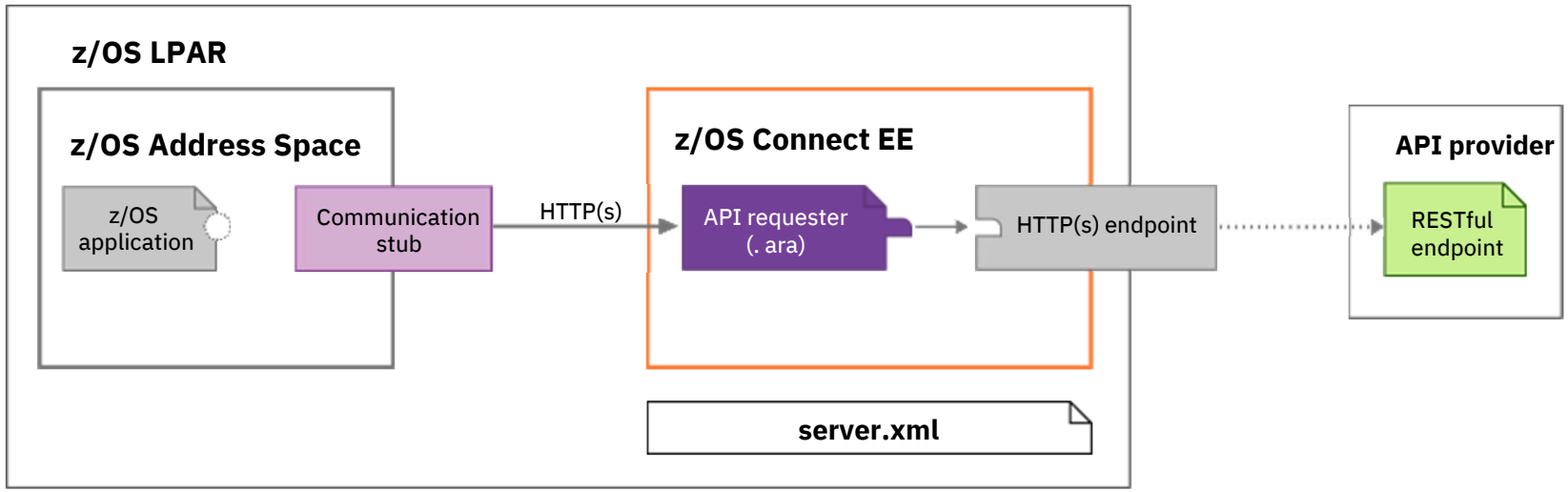
Options:

1. Basic Authentication
2. TLS Client / Server

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

© 2017, 2020 IBM Corporation

Encryption



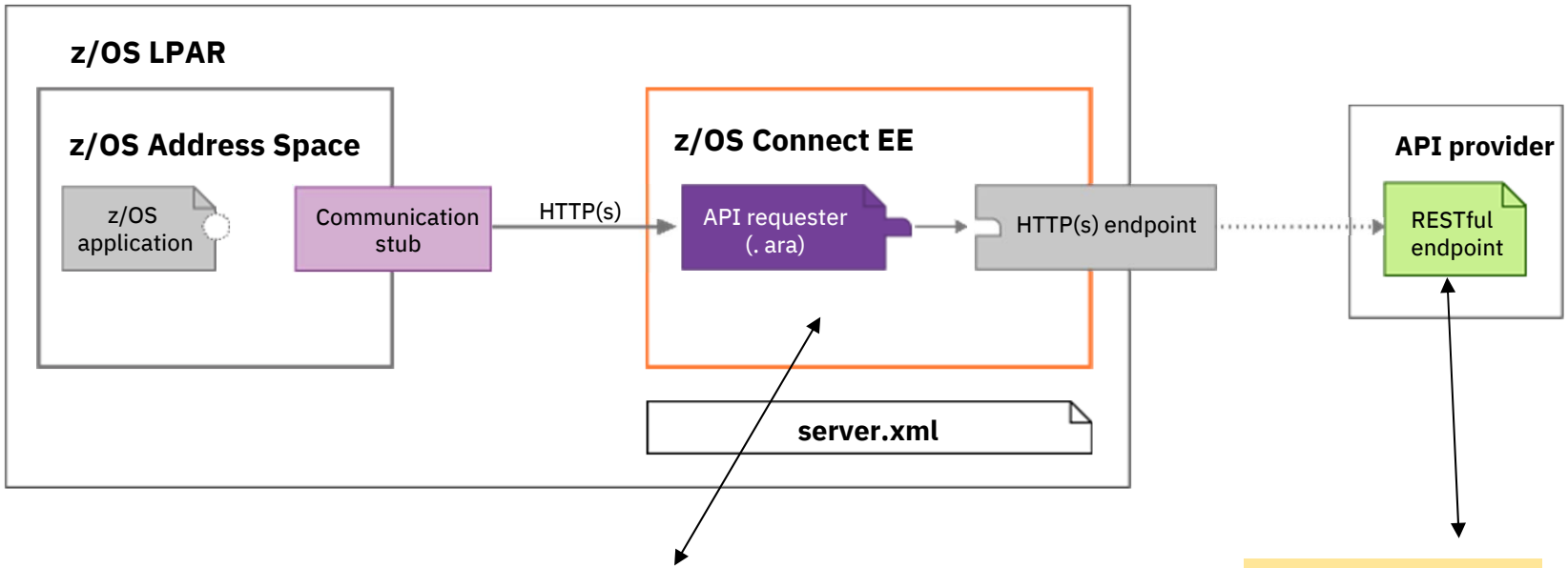
Options:

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

1. JSSE
2. AT-TLS

© 2017, 2020 IBM Corporation

Authorization



Authorization interceptor options:

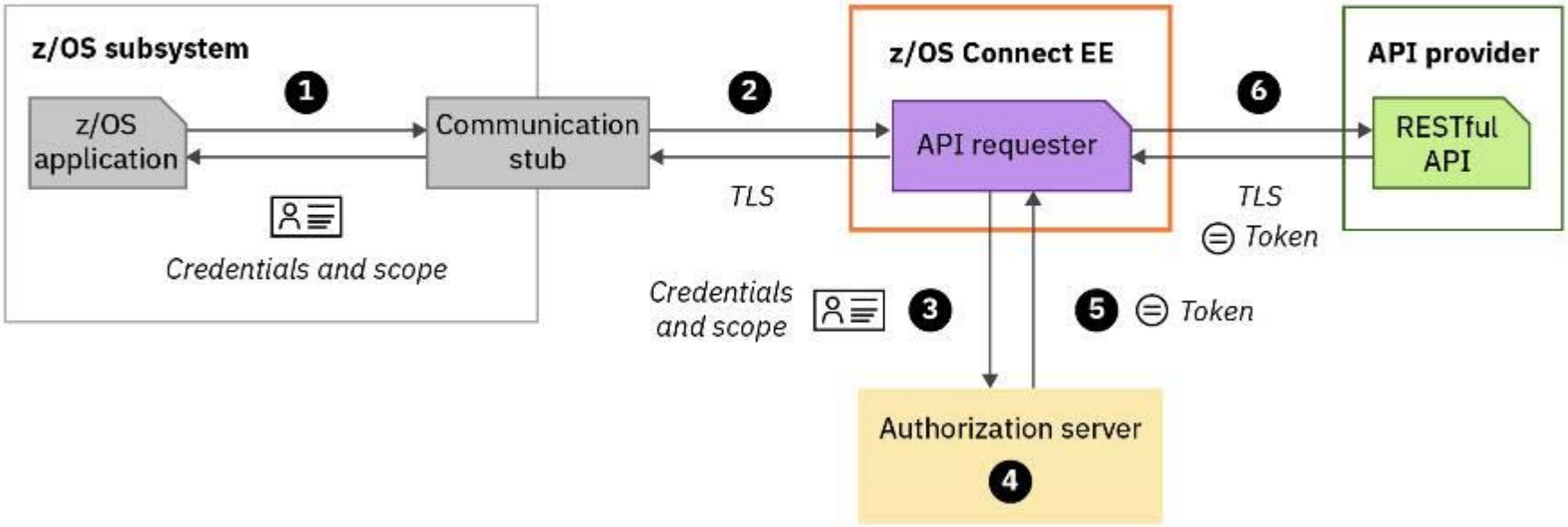
- Global level
- API requester level

OAuth or JWT access token

© 2017, 2020 IBM Corporation

Calling an API with OAuth 2.0 support

z/OS Connect EE



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

© 2017, 2020 IBM Corporation

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.

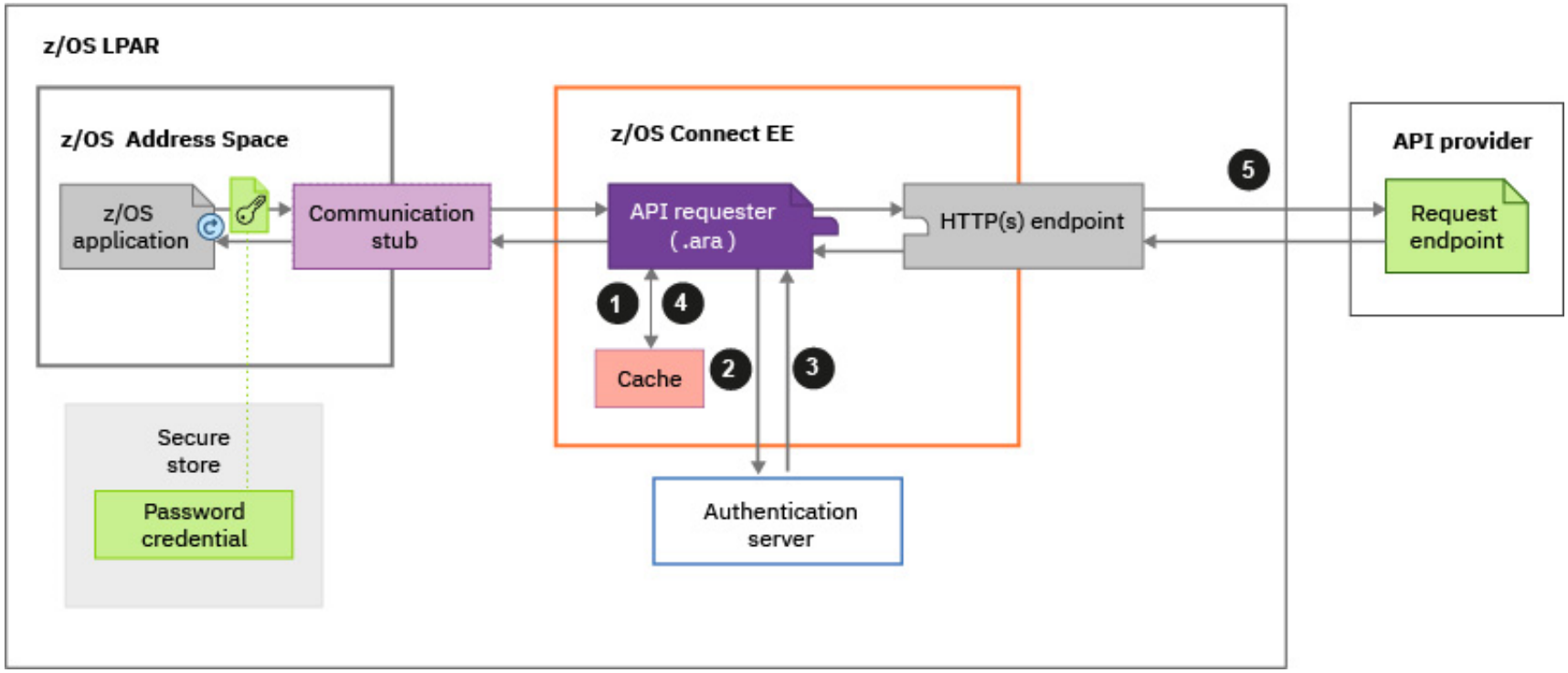
```
<zosconnect_endpointConnection id="orderDispatchAPI"
    host="https://154.2.45.123" port="443"
    authenticationConfigRef="myOAuthConfig"/>

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

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

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

optional

© 2017, 2020 IBM Corporation

Calling an API with JWT support

z/OS Connect EE



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

© 2017, 2020 IBM Corporation

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"/>
```


© 2017, 2020 IBM Corporation

Securing connection from z/OS Connect to API provider

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"/>
```


© 2017, 2020 IBM Corporation

Summary

© 2017, 2019 IBM Corporation

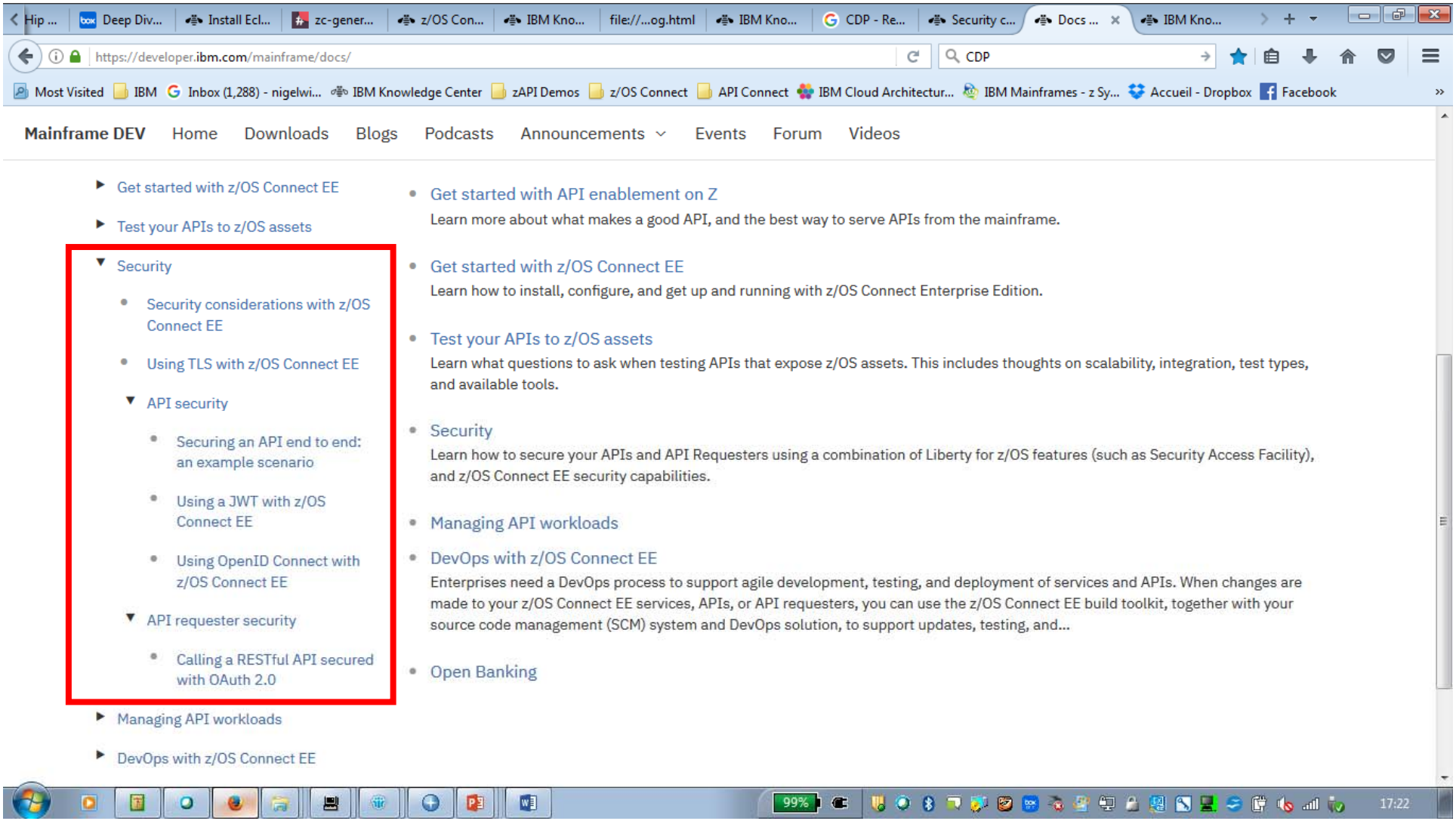
Summary

- 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

© 2017, 2020 IBM Corporation

More information

z/OS Connect EE



© 2017, 2020 IBM Corporation

/questions?thanks=true

Thank you for listening.

© 2018, 2019 IBM Corporation

© 2017, 2020 IBM Corporation