



# IBM z/OS Connect Enterprise Edition

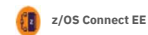
## Security

Mitch Johnson  
[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)  
Washington System Center



© 2017, 2019 IBM Corporation

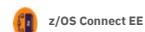
## Contents



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

© 2017, 2019 IBM Corporation

## General considerations for securing REST APIs

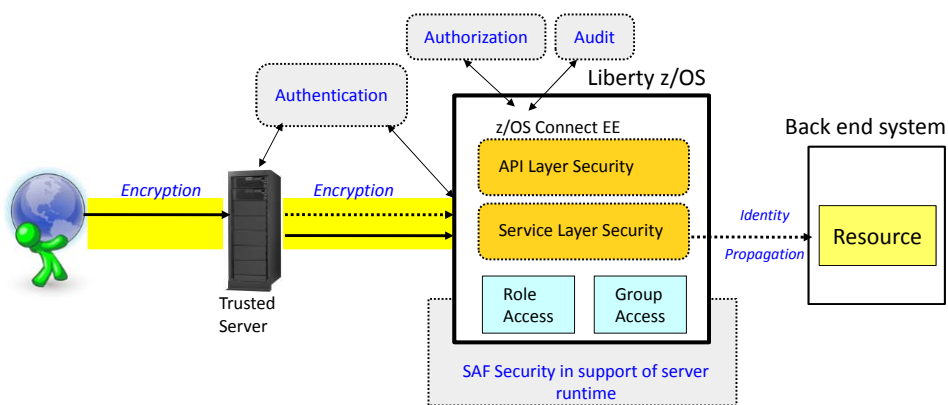
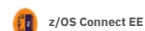


- Know who is invoking the API (**Authentication**)
- Ensure that the data has not been altered in transit (**Data Integrity**) and ensure confidentiality of data in transit (**Encryption**)
- Control access to APIs (**Authorization**)
  - End user
  - Application
- Know who invoked the APIs (**Audit**)



© 2017, 2019 IBM Corporation

## z/OS Connect EE API provider security overview

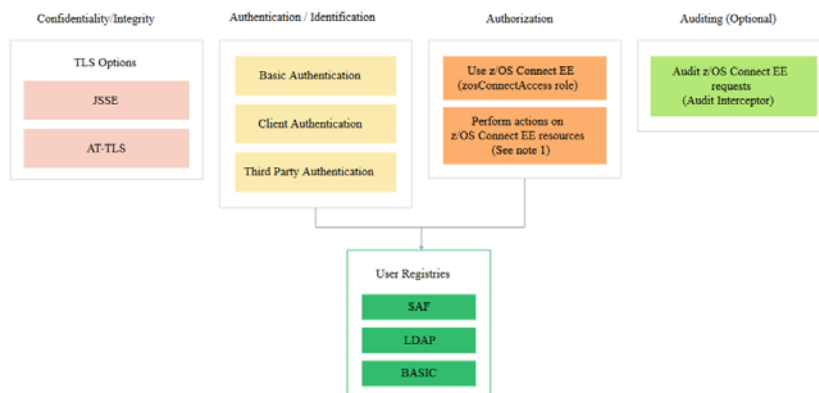


1. Authentication (basic, client certificates, 3<sup>rd</sup> party authentication)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (role and group access)
4. Audit
5. Configuring security with SAF
6. Back end identity propagation (CICS, IMS, D)

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

© 2017, 2019 IBM Corporation

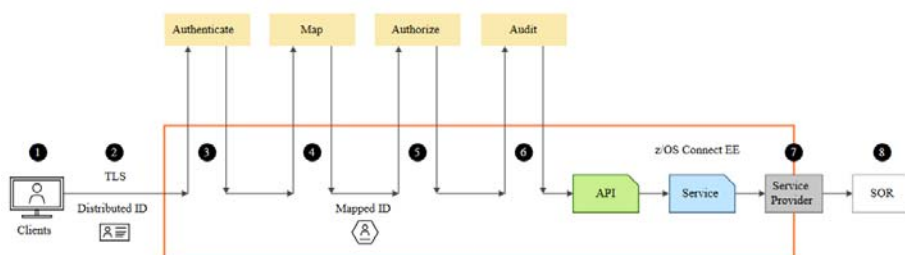
## z/OS Connect EE security options



<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, 2019 IBM Corporation

## 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

© 2017, 2019 IBM Corporation

## Security is configured in server.xml

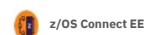


### Excerpt from server.xml

<pre>&lt;featureManager&gt;   &lt;feature&gt;appSecurity-2.0&lt;/feature&gt;   &lt;feature&gt;zosSecurity-1.0&lt;/feature&gt;   &lt;feature&gt;transportSecurity-1.0&lt;/feature&gt; &lt;/featureManager&gt;</pre>	Features
<pre>&lt;sslDefault sslRef="defaultSSLConfig"/&gt; &lt;ssl id="defaultSSLConfig"   keyStoreRef="defaultKeyStore"   outboundSSLRef="defaultKeyStore" /&gt;</pre>	SSL repertoire
<pre>&lt;keyStore id="defaultKeyStore" fileBased="false"   location="safkeyring:///Keyring.LIBERTY"   password="password" readOnly="true"   type="JCERACFKS" /&gt;</pre>	Key Store
<pre>&lt;webAppSecurity allowFailOverToBasicAuth="true" /&gt;</pre>	Authentication
<pre>&lt;safRegistry id="saf"/&gt; &lt;safAuthorization id="safAuth"/&gt; &lt;safCredentials profilePrefix="BBGZDFLT"   unauthenticatedUser="WSGUEST" /&gt;</pre>	Authorization

© 2017, 2019 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
- z/OS Connect security is implemented in many products including z/OS Connect, Liberty z/OS, SAF/RACF, CICS, IMS, DB2 ...
  - › 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, 2019 IBM Corporation



## Authentication

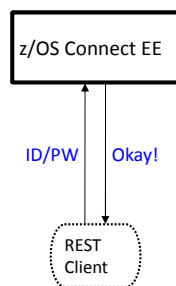
Obtaining an identity

© 2017, 2019 IBM Corporation

## Authentication

Several different ways this can be accomplished:

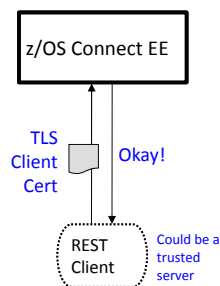
### Basic Authentication



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

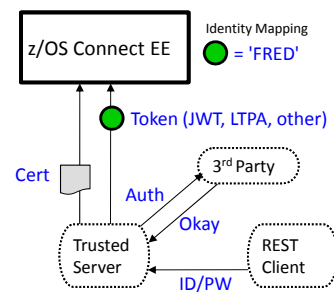
- Basic (server.xml)
- LDAP
- SAF

### Client Certificate



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

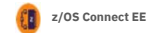
### Third Party Authentication



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

© 2017, 2019 IBM Corporation

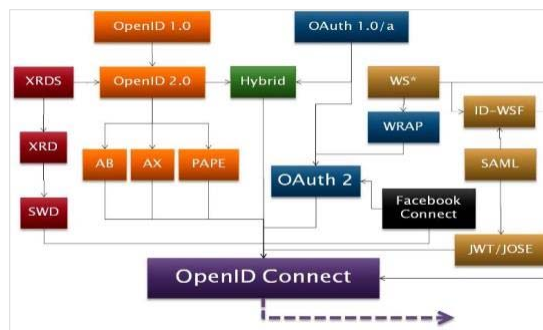
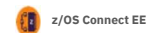
## Security token types by z/OS Connect EE



Token type	How used	Pros	Cons
LTPA	Authentication technology used in IBM WebSphere	<ul style="list-style-type: none"> <li>Easy to use with WebSphere and DataPower</li> </ul>	<ul style="list-style-type: none"> <li>IBM Proprietary token</li> </ul>
SAML	XML-based security token and set of profiles	<ul style="list-style-type: none"> <li>Token includes user id and claims</li> <li>Used widely with SoR applications</li> </ul>	<ul style="list-style-type: none"> <li>Tokens can be heavy to process</li> <li>No refresh token</li> </ul>
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	<ul style="list-style-type: none"> <li>Used widely for SoE applications e.g with Google, Facebook, Microsoft, Twitter ...</li> </ul>	<ul style="list-style-type: none"> <li>Needs introspection endpoint to validate token</li> </ul>
JWT	JSON security token format	<ul style="list-style-type: none"> <li>More compact than SAML</li> <li>Ease of client-side processing especially mobile</li> </ul>	

© 2017, 2019 IBM Corporation

## Open security standards




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

© 2017, 2019 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



z/OS Connect EE

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

© 2017, 2019 IBM Corporation

## Why JWT with z/OS Connect EE?

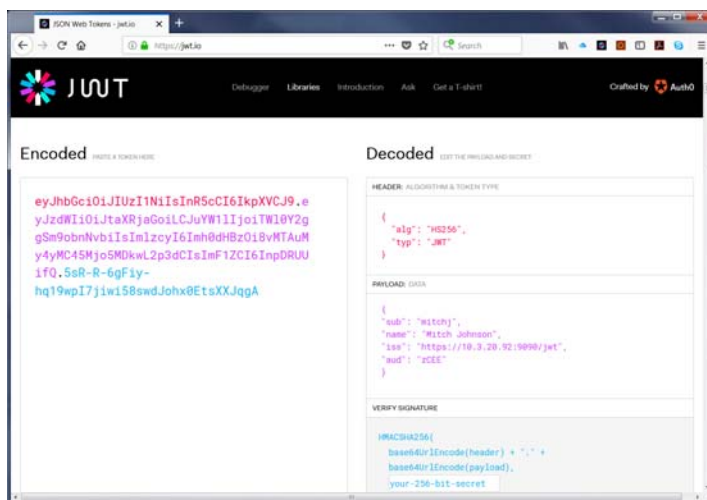
- Token validation does **not** require an additional trip and can be validated locally by z/OS Connect server
- Parties can easily agree on a specific set of **custom** claims in order to exchange both authentication and authorization information
- Widely adopted by different Single Sign-On solutions 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, 2019 IBM Corporation

## JWT (JSON Web Token)

z/OS Connect EE

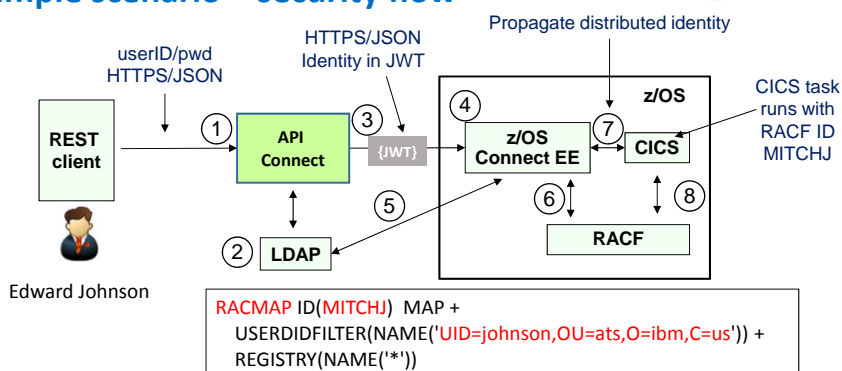
- JWT is a compact way of representing claims that are to be transferred between two parties
- Normally transmitted via HTTP header
- Consists of three parts
  - Header
  - Payload
  - Signature



© 2017, 2019 IBM Corporation

## Example scenario – security flow

z/OS Connect EE

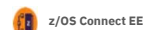


1. User authenticates with the managed API using a "distributed" identity and a password
2. LDAP is used as the user registry for distributed users and groups
3. API Connect generates a JWT and forwards the token with the request to z/OS Connect EE
4. z/OS Connect EE validates JWT
5. z/OS Connect EE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

© 2017, 2019 IBM Corporation



## JWT used in scenario



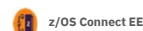
```
{
  "alg": "RS256"
  "typ": "JWT"
}
{
  "sub": "johnson",
  "iss": " https://10.3.20.92:9090/jwt",
  "aud": "zCEE"
}
```

- 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 **johnson** identifies the principal that is the subject of the JWT
- The **aud** (audience) claim **zCEE** identifies the recipients that the JWT is intended for (stands for

© 2017, 2019 IBM Corporation

IBM Client Center Montpellier

## 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="RS" clientId="RS-JWT-ZCEE" inboundPropagation="required"
  signatureAlgorithm="RS256" trustStoreRef="JWTTrustStore"
  trustAliasName="JWTapicSign" userIdentityToCreateSubject="sub"
  mapIdentityToRegistryUser="true"
  issuerIdentifier="https://10.3.20.92:9090/jwt" authnSessionDisabled="true"
  audiences="zCEE"/>
```

- **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, 2019 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="RS" clientId="RS-JWT-ZCEE" inboundPropagation="required"
signatureAlgorithm="RS256" trustStoreRef="JWTTrustStore"
trustAliasName="JWTapicSign" userIdentityToCreateSubject="sub"
mapIdentityToRegistryUser="true" issuerIdentifier="https://10.3.20.92:9090/jwt"
authnSessionDisabled="true" audiences="zCEE" authFilterRef="API Gateway"/>

<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>
```

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.

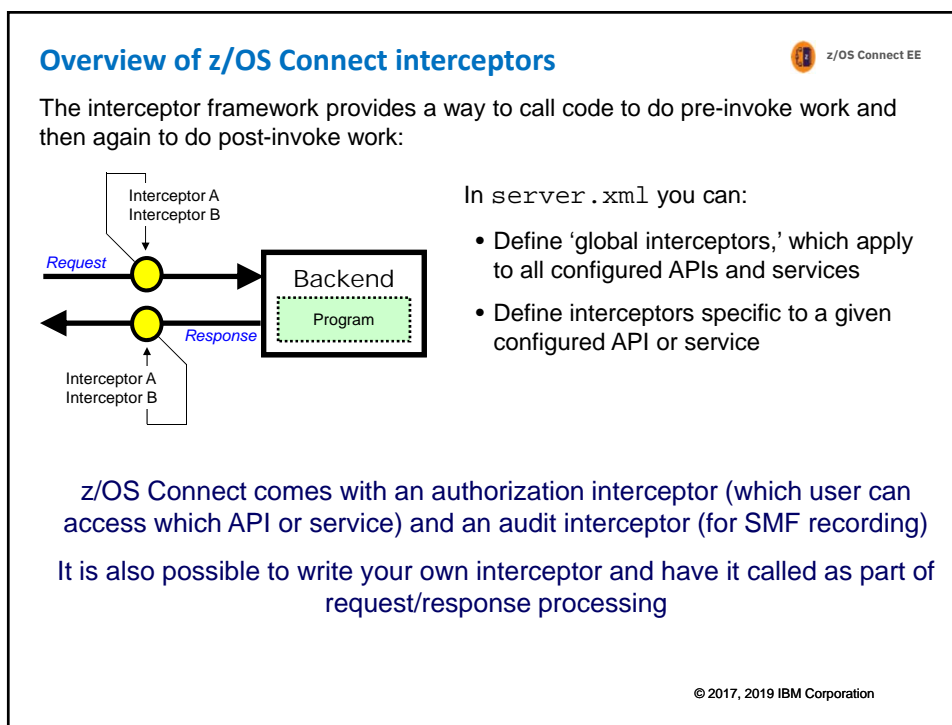
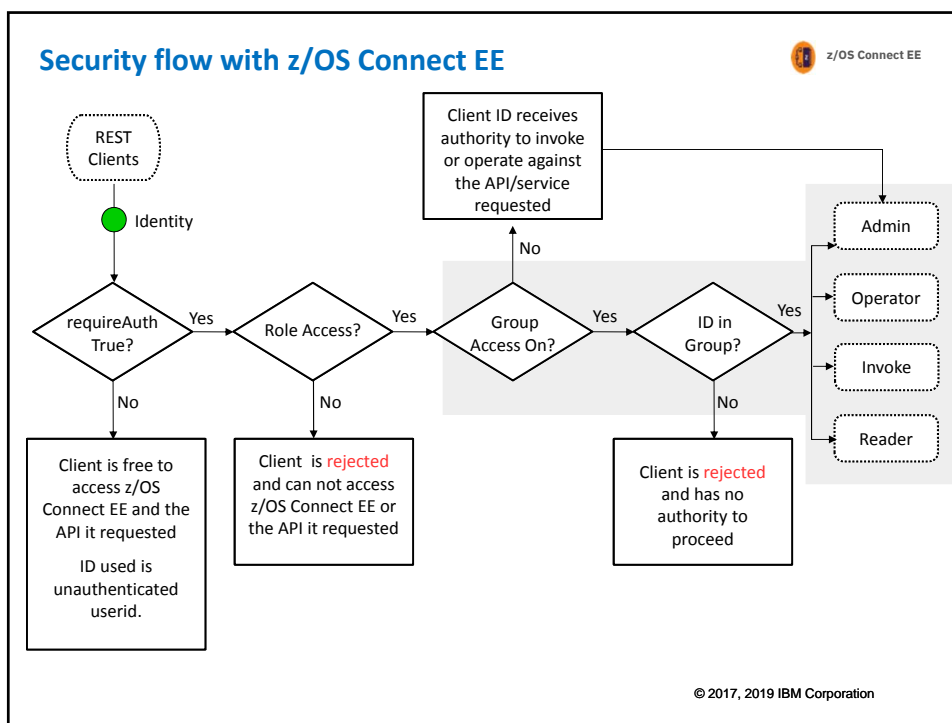
© 2017, 2019 IBM Corporation

 z/OS Connect EE

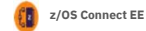
## Authorization

Once we have an identity

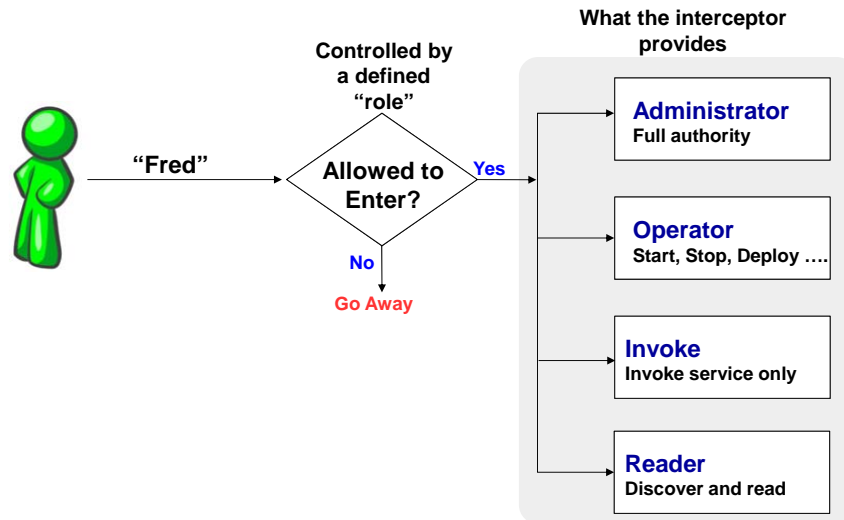
© 2017, 2019 IBM Corporation



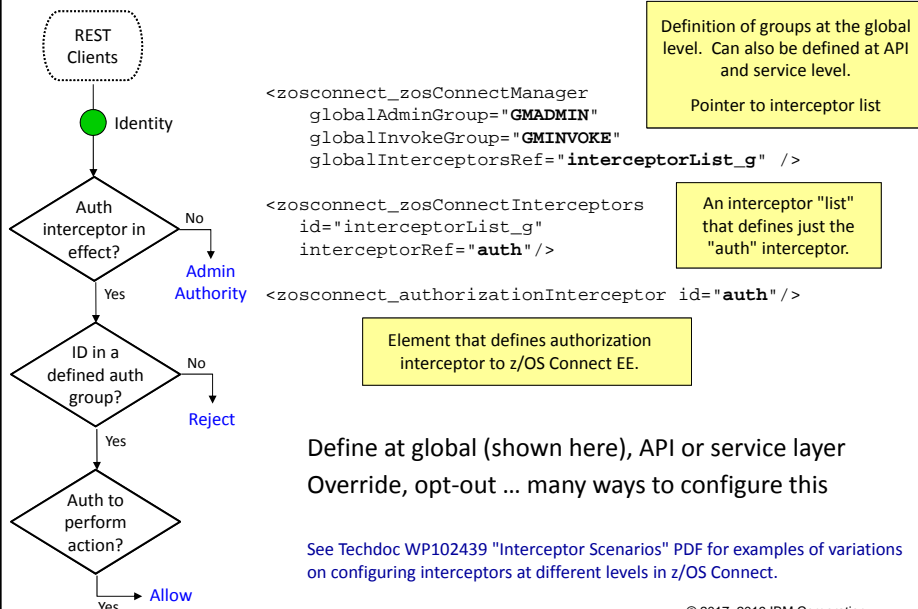
## Authorization interceptor



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:

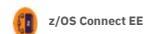


## Configuring authorization



© 2017, 2019 IBM Corporation

## Configuring interceptors - Example



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.

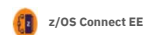
```
<zoscconnect_zosConnectManager globalInterceptorsRef="interceptorList_g"
globalAdminGroup="GMADMIN" globalInvokeGroup="GMINVOKE"/>

<zoscconnect_authorizationInterceptor id="auth"/>
<zoscconnect_auditInterceptor id="audit"/>

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

<zoscconnect_zosConnectAPIs location="">
  <zoscConnectAPI name="catalog" interceptorsRef="interceptorList_s" />
</zoscconnect_zosConnectAPIs>
```

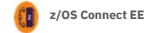
© 2017, 2019 IBM Corporation



## 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

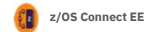
*Server Identification  
Section*

- 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*

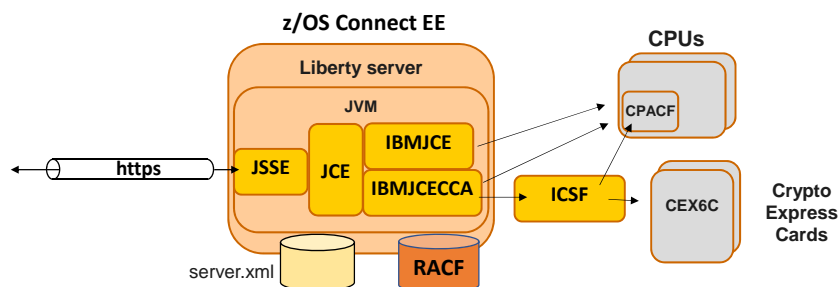
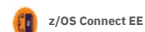
© 2017, 2019 IBM Corporation

## Encryption



© 2017, 2019 IBM Corporation

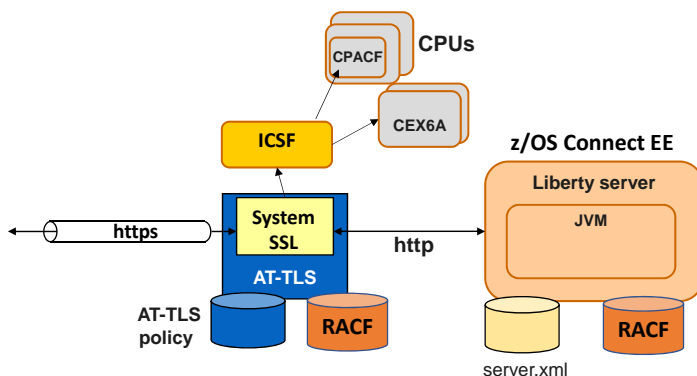
## Using JSSE with z/OS Connect EE



- 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**

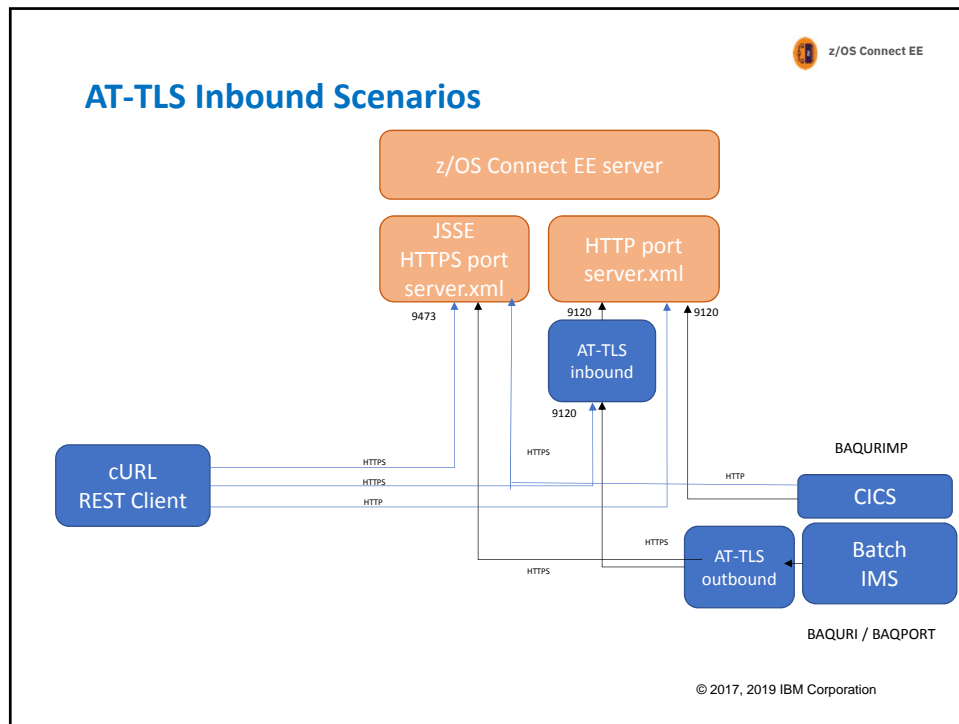
© 2017, 2019 IBM Corporation

## 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, 2019 IBM Corporation



z/OS Connect EE

### JSSE and AT-TLS comparison

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	<b>No</b>
Support for requireSecure option on APIs	Requires that API requests are sent over HTTPS	Yes	<b>No</b>
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	<b>No</b>	Yes
zIIP processing	Offload TLS processing to zIIP	Yes	<b>No</b>
CPACF	Offload symmetric encryption to CPACF	Yes	Yes
CEX6	Offload asymmetric operations to Crypto Express cards	Yes	Yes

© 2017, 2019 IBM Corporation



## Configuring TLS Encryption with JSSE

© 2017, 2019 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, 2019 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, 2019 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

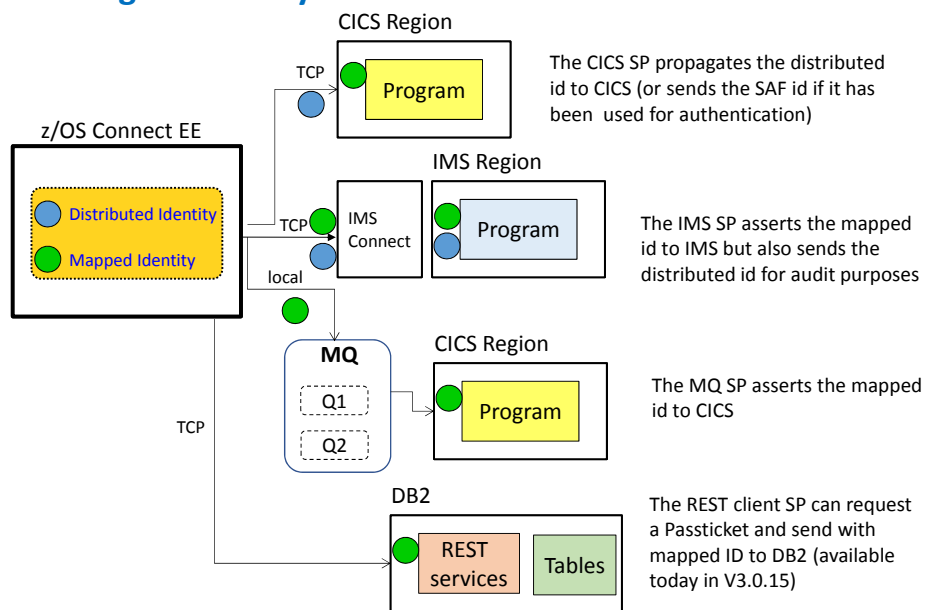
© 2017, 2019 IBM Corporation

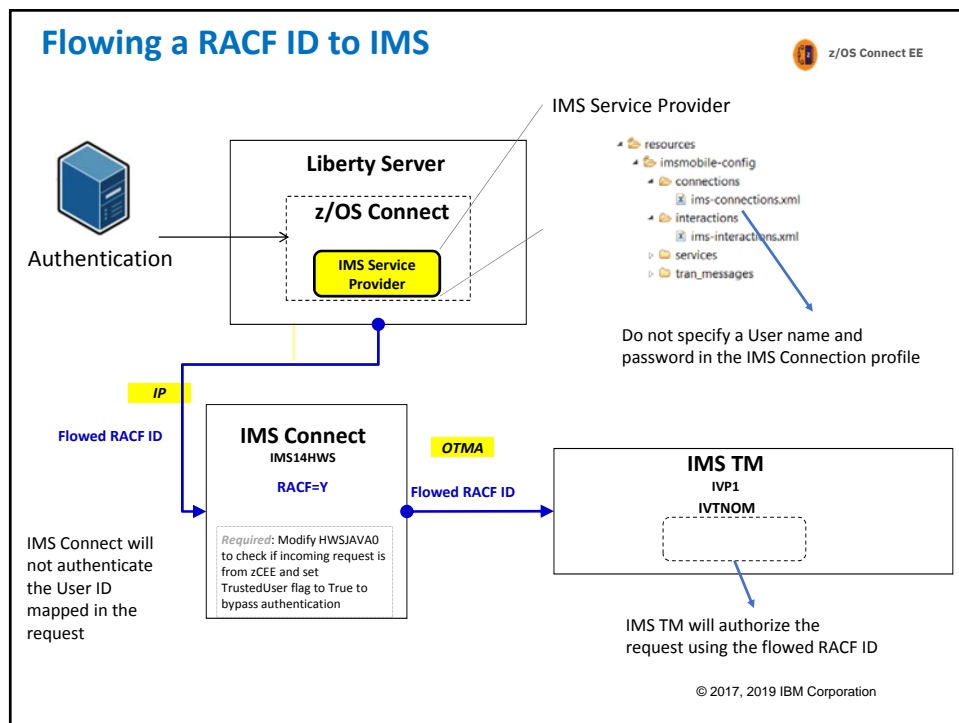
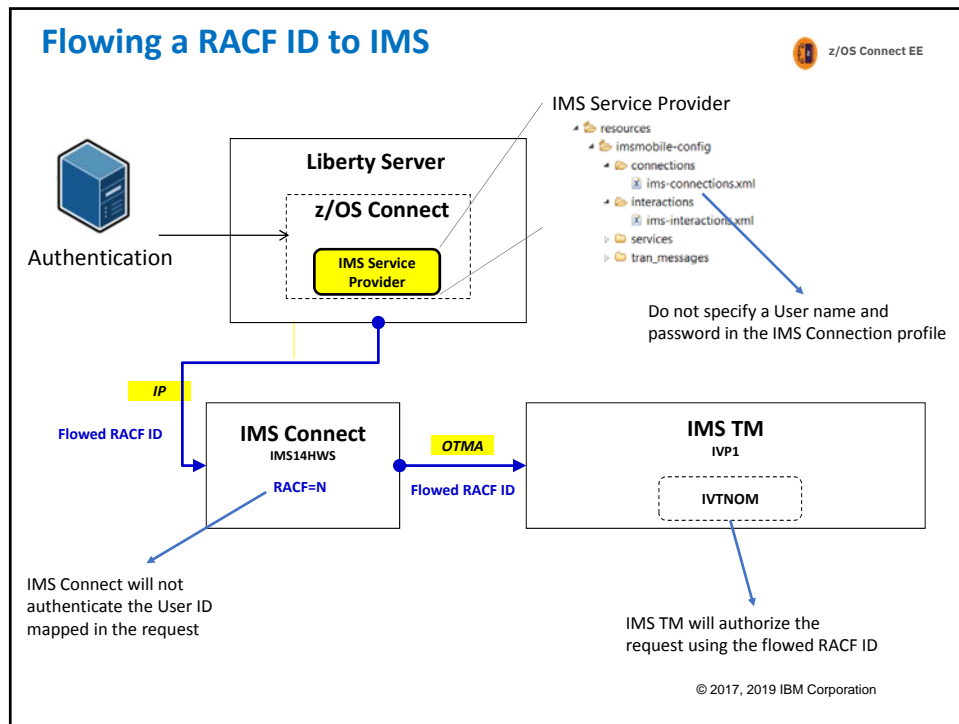


## Flowing identities to back end systems

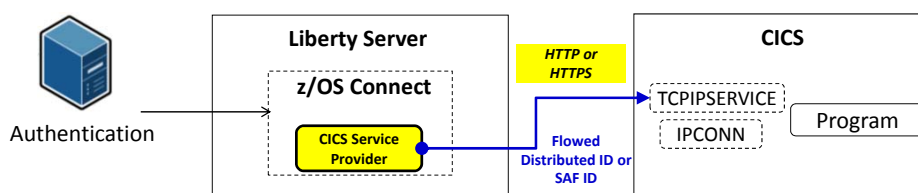
© 2017, 2019 IBM Corporation

### Flowing an identity to the back end





## Flowing a user ID with CICS service provider



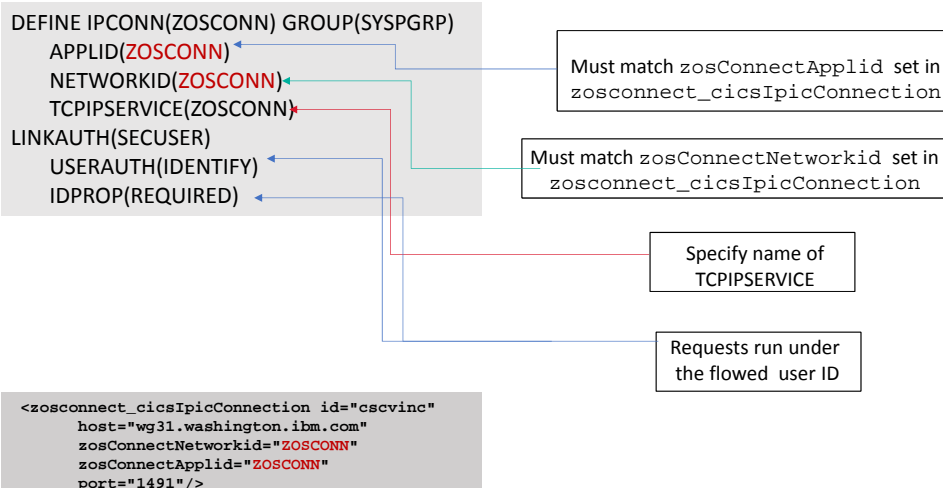
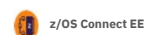
IPIC connections enforce **bind** security to prevent an unauthorized client system from connecting to CICS, **link** security to restrict the resources that can be accessed over a connection to a CICS system, and **user** security to restrict the CICS resources that can be accessed by a user

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.

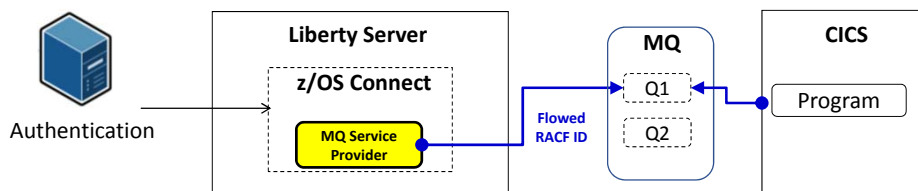
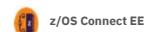
© 2017, 2019 IBM Corporation

## CICS IPCONN



© 2017, 2019 IBM Corporation

## Flowing a user ID with MQ service provider

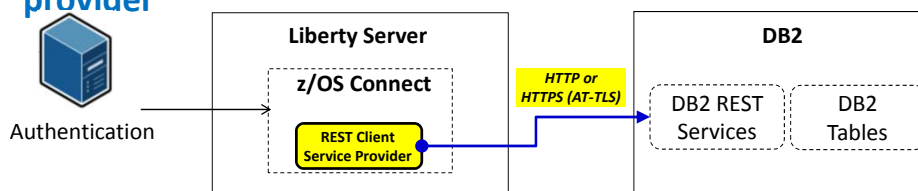


Configuration attributes on the `mqzOSConnectService` element, and the `properties.wmqJMS` sub-element of the `jmsConnectFactory` element affect which user ID and optional password are presented to the queue manager.

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

© 2017, 2019 IBM Corporation

## Setting the user ID for the REST client service provider



Authentication options:

1. User ID / password
2. TLS Client Certificate
3. Passticket support (In Open Beta currently)

```
<zoscconnect_zosConnectServiceRestClientBasicAuth
```

```
...
userName="EMPLOY1"
```

```
password="{xor}GhIPExAGDwg="/>
```

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

```
<zoscconnect_zosConnectServiceRestClientConnection
```

```
...
sslCertsRef="sslCertificates"/>
```

TLS client authentication

```
<zoscconnect_zosConnectServiceRestClientBasicAuth
```

```
...
applName="applName"/>
```

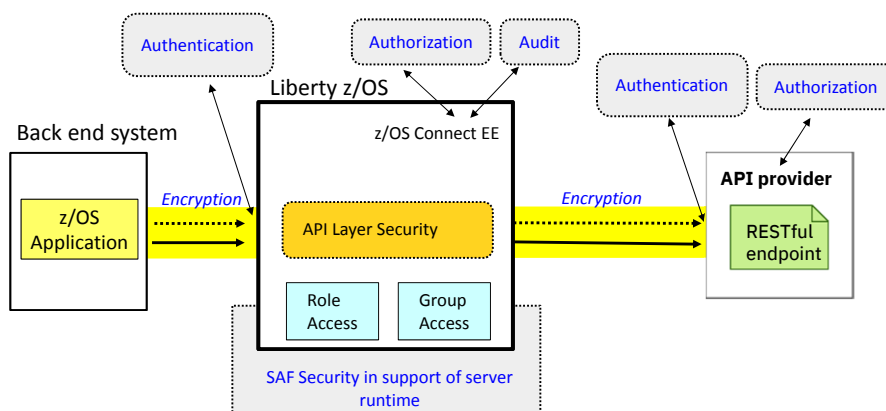
z/OS Connect requests a PassTicket from RACF

© 2017, 2019 IBM Corporation

## What's different for API Requester?

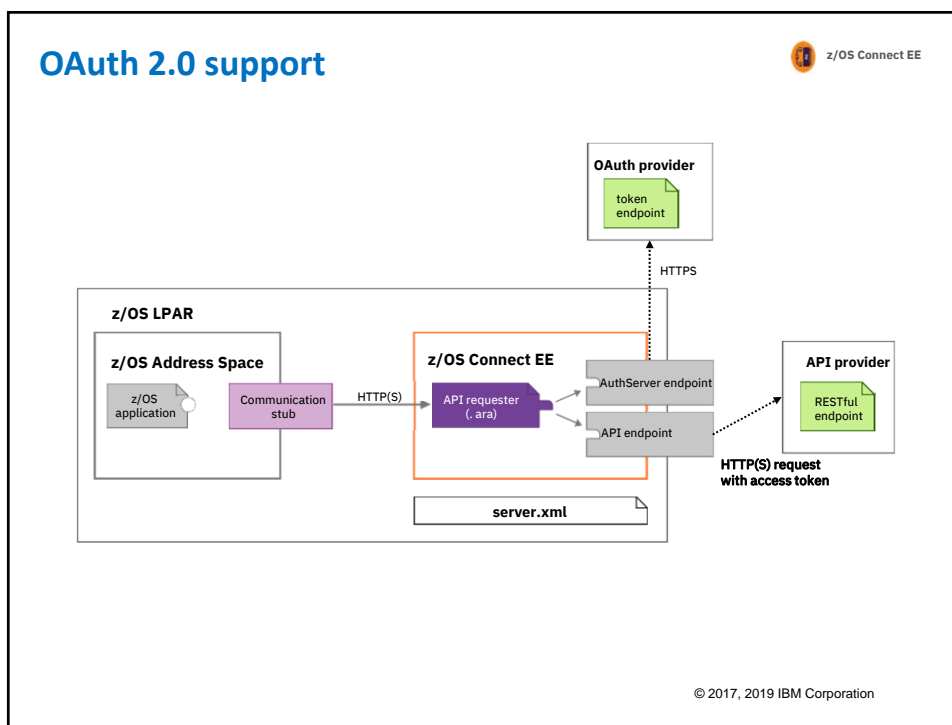
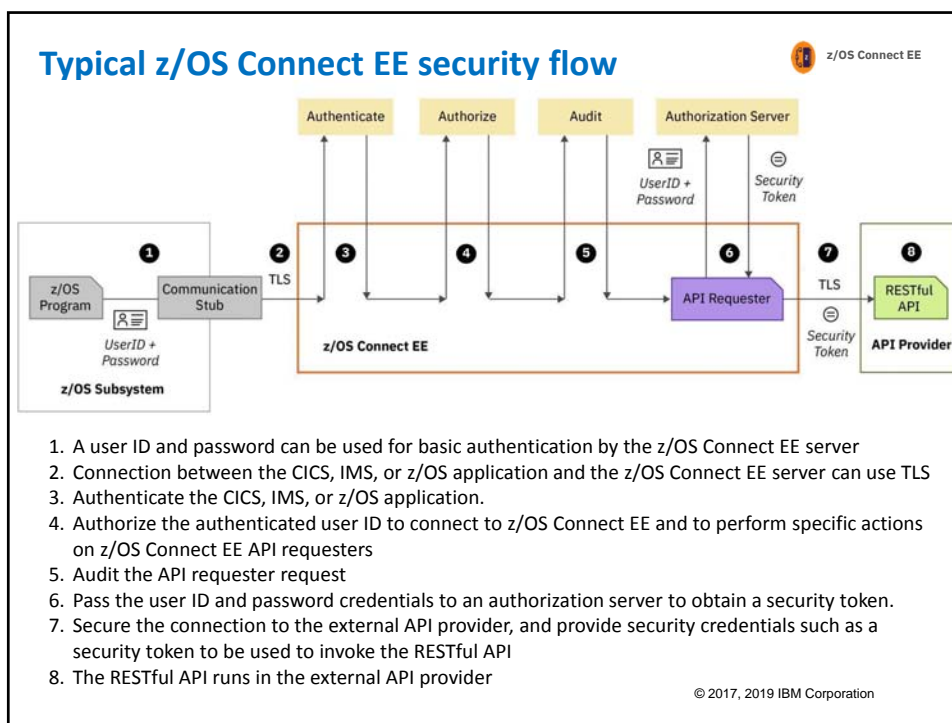
© 2017, 2019 IBM Corporation

### API requester security – overview



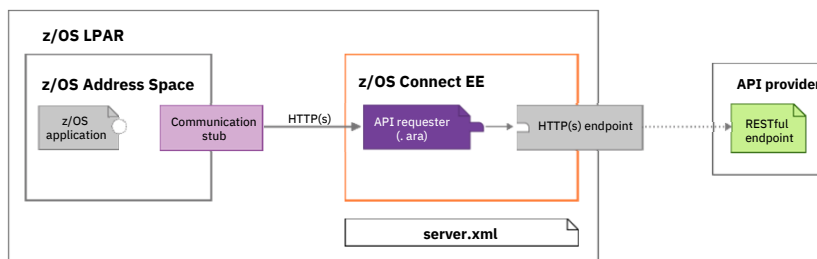
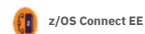
1. Authentication (basic, client certificate)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (OAuth)
4. Audit
5. Configuring security with SAF

© 2017, 2019 IBM Corporation





## Encryption



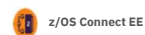
Options:

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

1. JSSE
2. AT-TLS

© 2017, 2019 IBM Corporation

## Configuring OAuth support



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" />

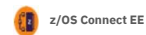
```

© 2017, 2019 IBM Corporation



## Summary

© 2017, 2019 IBM Corporation



## Summary

- Define clear security requirements before deciding on a security design
- Security design needs to consider
  - Authentication
  - Encryption
  - Authorization
  - Audit
  - Protection against attack
  - Rate limiting
- 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 the API Gateway

© 2017, 2019 IBM Corporation

## More information

z/OS Connect EE

Most Visited IBM G Inbox (2,288) - nigelw... IBM Knowledge Center z/OS Connect API Connect IBM Cloud Architectur... IBM Mainframes - z Sy... Accueil - Dropbox Facebook

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

© 2017, 2019 IBM Corporation

