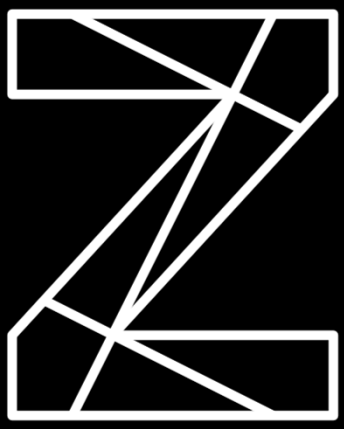
z/OS Connect Enterprise Edition

Deployment Planning Considerations



IBM z/OS Connect Enterprise Edition (z/OS Connect EE) is IBM's strategic solution for API enablement of z/OS applications and data. Perhaps you are one of the many clients that have created, deployed and tested your initial APIs using z/OS Connect EE. You like the ease with which APIs can be created but you're wondering about some of the non-functional requirements:

- How to deploy APIs in production.
- How to secure APIs.
- How to make sure that the APIs are always available.
- How to monitor APIs.
- How to debug problems.
- What about performance.

This presentation is intended for Architects, Systems Programmers and API Developers who are involved in planning the deployment of IBM z/OS Connect Enterprise Edition V3.0 (z/OS Connect EE V3.0).

It is assumed that the audience is familiar with the concepts and operation of z/OS Connect EE. The focus here is on deployment of z/OS Connect EE.

The current version of this presentation can be downloaded here:
http://ibm.biz/zcee-deployment-guide

Important Disclaimer

IBM's statements regarding its plans, directions and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

/agenda

Agenda

Quick intro to z/OS Connect EE

Common scenarios

DevOps
- What is DevOps?
- DevOps pipeline for z/OS Connect EE
- Building
- Testing
- Deploying

Security
- Security options
- API provider security flow
- API requester security flow
- Confidentiality / Integrity
- Authentication / Identification
- Authorization
- Auditing

Workload management
- High availability configuration
- Policy-based API processing

Monitoring
- Real time monitoring
- End-to-end transaction tracking
- Operation analytics

Problem determination
- Messages and trace
- Debugging

Performance
- Main factors that impact performance
- Classifying API requests with WLM
- Measuring API requests with RMF
- Example performance test

More information

/zos_connect_ee

Truly RESTful APIs to and from your mainframe.

The focus for this presentation is on deployment of z/OS Connect EE. But let's first review some of the capabilities of z/OS Connect EE.

Use API provider **to expose a z/OS asset**

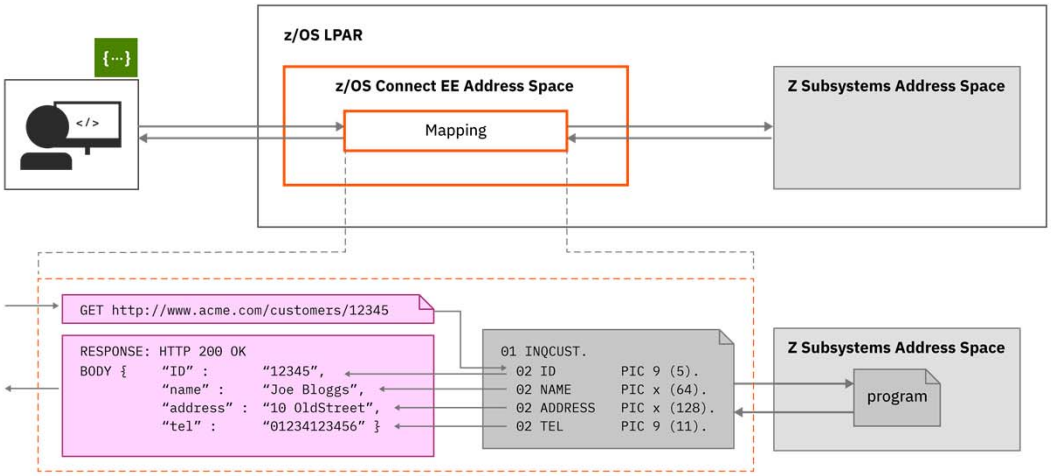
z/OS Connect EE

z/OS Connect EE provides a common entry point for REST HTTP calls to reach business assets and data on z/OS operating systems. Where these assets run is specified in the z/OS Connect EE configuration, which relieves client applications in the cloud, mobile, and web worlds of the need to understand the details about how to reach them and how to convert payloads to and from the formats that the applications require. APIs can be enabled without writing code and tooling is provided for creating the data transformation artifacts.

With z/OS Connect EE, mobile and cloud application developers can incorporate z/OS data and transactions into their applications, whether they work inside or outside the enterprise, without needing to understand z/OS subsystems. The z/OS resources appear as any other REST API. This capability is referred to as the ***API provider support***.

Data mapping

z/OS Connect EE



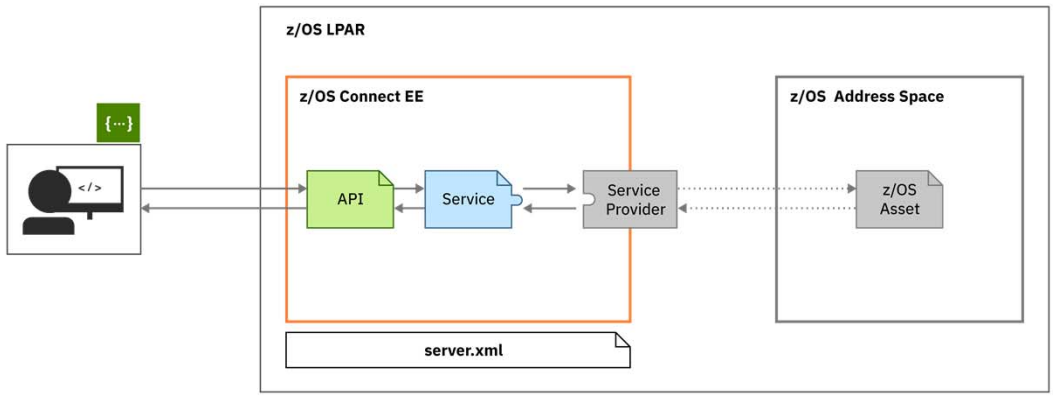
A key role of z/OS Connect EE is the mapping of REST/JSON messages to and from the message formats expected by z/OS applications.

In this simple example, we see a mapping between a binary data structure represented in a COBOL form and a JSON response message.

We also see the REST API request and a mapping of a URI path parameter to a field in a COBOL copybook. z/OS Connect EE provides granular mapping of data structures, which makes it possible to create *genuinely* RESTful APIs.

Expose a z/OS asset

z/OS Connect EE



This chart shows the different z/OS Connect EE components that are involved in exposing a z/OS asset as an API:

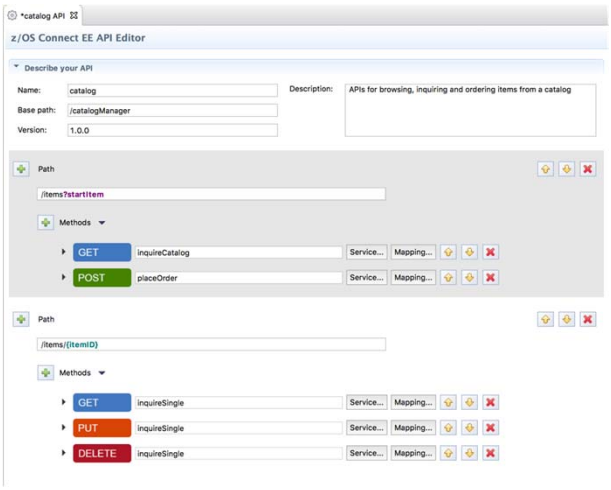
- A **service** in z/OS Connect EE is used by a REST API to act on a z/OS resource through connections and data transformation functions provided through a service provider. Information about the service is contained in a service archive (.sar) file and includes information about the request and response JSON schemas required by the service.
- For z/OS Connect EE services, you can create REST **APIs** that define how an HTTP action such as GET, PUT, POST or DELETE would act on the services. Information about the REST APIs for a service is contained in an API archive (.aar) file that can be deployed to z/OS Connect EE.
- The **service provider** forwards requests to the z/OS address space that hosts the z/OS asset, for example, a CICS region.

To learn more about the service and API creation workflow see:
https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/overview/api_design_workflow.html

API toolkit

z/OS Connect EE

API definition



The API toolkit is designed to encourage RESTful API design.

Once you define your API, you can map backend services to each request.

Your services are represented by `.sar` files, which you import into the API toolkit.

The z/OS Connect EE API toolkit is an Eclipse-based workstation tool that you install into IBM Explorer for z/OS to create services and REST APIs for accessing z/OS resources.

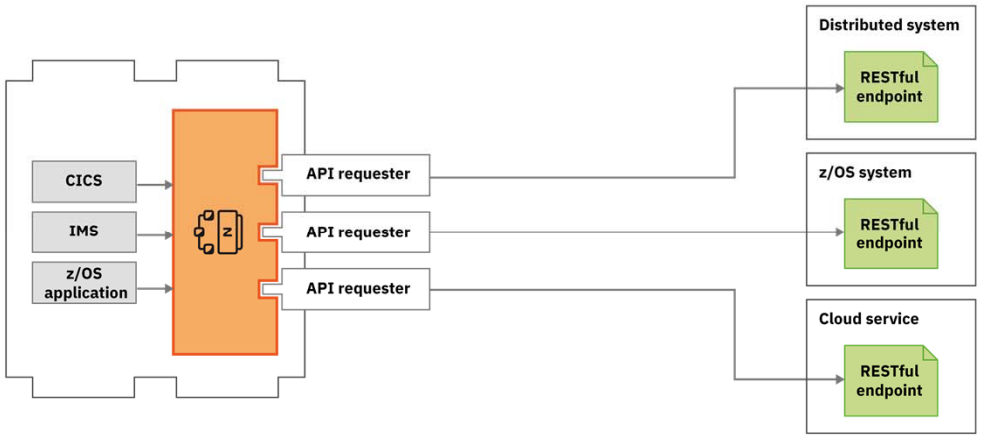
In the API toolkit, you can create two types of projects: service projects and API projects. This chart is showing an API project.

The API Editor (part of the API toolkit) is laid out in a way that encourages you to create an API that conforms to the REST architectural style.

A backend service is represented in the editor as a .sar file, that you can generate using the provided tooling. You can create a CICS, IMS, Db2 and IBM MQ services in a z/OS Connect EE service project.

Use API requester **to call external APIs from z/OS assets**

z/OS Connect EE



z/OS Connect EE also provides the capability that allows z/OS-based programs to access any RESTful endpoint, inside or outside the enterprise, for example a cloud-based microservice. This framework enables CICS, IMS and other z/OS applications to call RESTful APIs through z/OS Connect EE. This capability is referred to as the ***API requester support***.

Learn more about the API requester support in the z/OS Connect EE Knowledge Center: **ibm.biz/zosconnect-api-requester-overview**

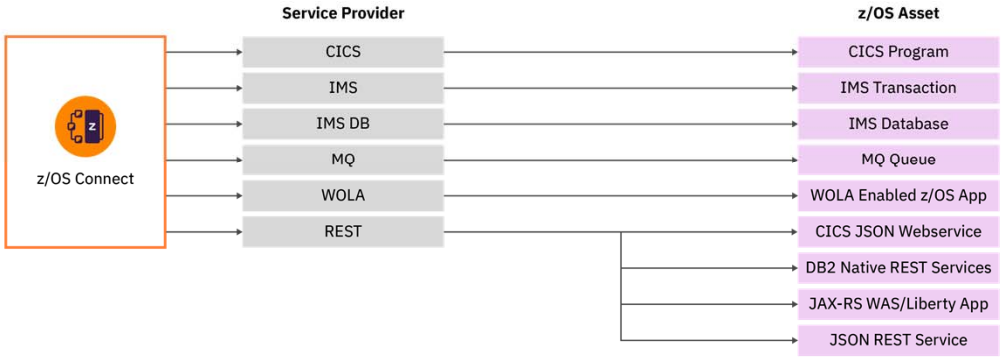
/common_scenarios

Typical connection patterns to different subsystems.

What assets can z/OS Connect EE map to?

z/OS Connect EE

And which service provider should I use?



The core service providers included with z/OS Connect EE provide API access to a wide range of z/OS assets.

A z/OS Connect EE service provider forwards requests to a System of Record (SoR). The following service providers are included with z/OS Connect EE:

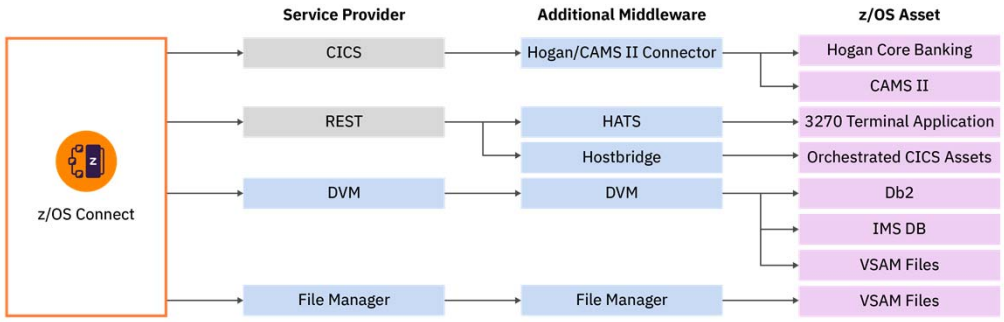
- CICS
- IMS
- IMS Database
- IBM MQ
- REST client
- WOLA (this service provider can be used by both WOLA-enabled applications and CICS to support z/OS Connect EE V2 configurations).

Alternatively you can use a service provider that is supplied with the SoR to plug into the framework, or you can write your own. All service providers must implement the *com.ibm.zosconnect.spi.Service* SPI.

z/OS Connect EE 3rd party integrations

z/OS Connect EE

Additional value from the ecosystem



z/OS Connect EE is pluggable and extensible allowing 3rd Party Service Providers to expand the list of z/OS assets you can expose as APIs

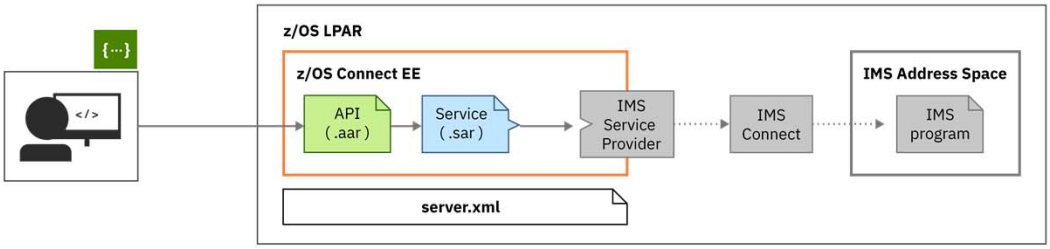
There is an ecosystem of 3rd party plugins that extend the types of z/OS assets that can be API enabled using z/OS Connect EE:

- The CICS service provider can be used to connect to Hogan Core Banking applications
- 3270 terminal applications can be accessed by using the REST client service provider to call Host Access Transformation Services (HATS) JSON Services
- The REST client service provider can be used with Hostbridge to connect to CICS programs or 3270 terminal applications. See **https://www.hostbridge.com/hostbridge-and-ibm-zcee-perfect-tag-team/**
- The Data Virtualisation Manger (DVM) service provider can be used for accessing Db2, IMS DB and VSAM Data. See **https://www.ibm.com/support/knowledgecenter/SS4NKG_1.1.0/havuga10/topics/dvs_sg_con_zos_connect.html**
- The File Manager service provider can be used for connecting to VSAM files. See **https://www.ibm.com/support/knowledgecenter/SSXJAV_14.1.0/com.ibm.filemanager.doc_14.1/base/fmsp-intro.html**

Common scenario - connect to IMS

z/OS Connect EE

Topology



Configure the connection to IMS through `ims-connections.xml` and `ims-interactions.xml` in the IMS service registry.

ibm.biz/zosconnect-scenarios

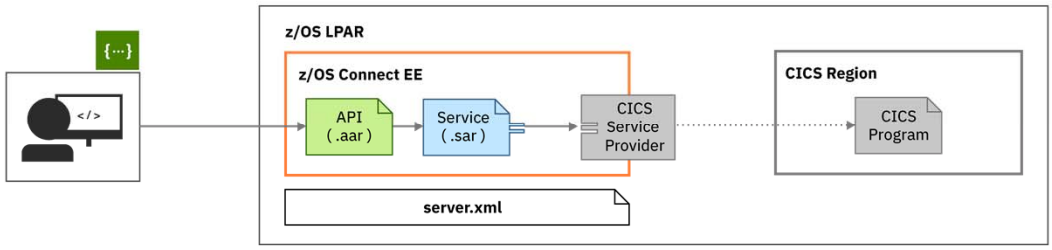
The IMS service provider connects to IMS through IMS Connect.

The IMS service registry contains the connection profiles and interaction profiles for IMS services. It is installed into a location that is defined in the server.xml file during initial configuration of a server instance based on the imsmobile templates.

Common scenario - connect to CICS

z/OS Connect EE

Topology



Connection to CICS is configured in `server.xml`.

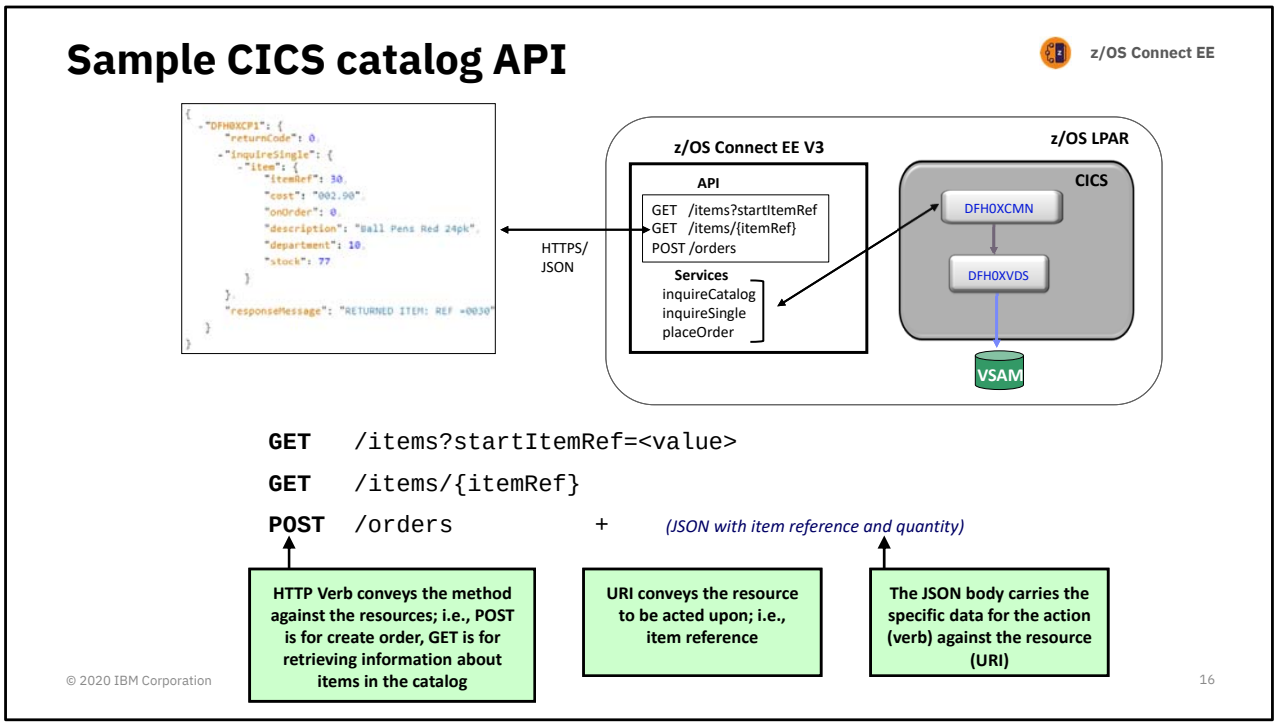
An IPIC connection must be configured in CICS.

ibm.biz/zosconnect-scenarios

This CICS service provider that is supplied with z/OS Connect EE connects to CICS using an IPIC connection which is configured in server.xml.

Both COMMAREA and channels with multiple containers are supported.

You can set the CICS transaction ID on the connection for all services. Furthermore, individual services can override the connection's transaction ID with a transaction ID in the service archive file, or in the server.xml configuration file using the service element under the *zosconnect_services* element.



This chart shows a sample CICS catalog API.

For listing the catalog we use an HTTP GET request. The URI specifies a collection of items. A query parameter is used to identify where in the catalog we want to start reading from. The JSON response message will contain an array of items and other information including the number of items returned. There is no JSON request message; everything is specified in the URI.

We also use a GET request to retrieve the details of an item. A path parameter is used to specify the item reference number. The JSON response message will contain the details of the item (for example the price, number in stock etc.)

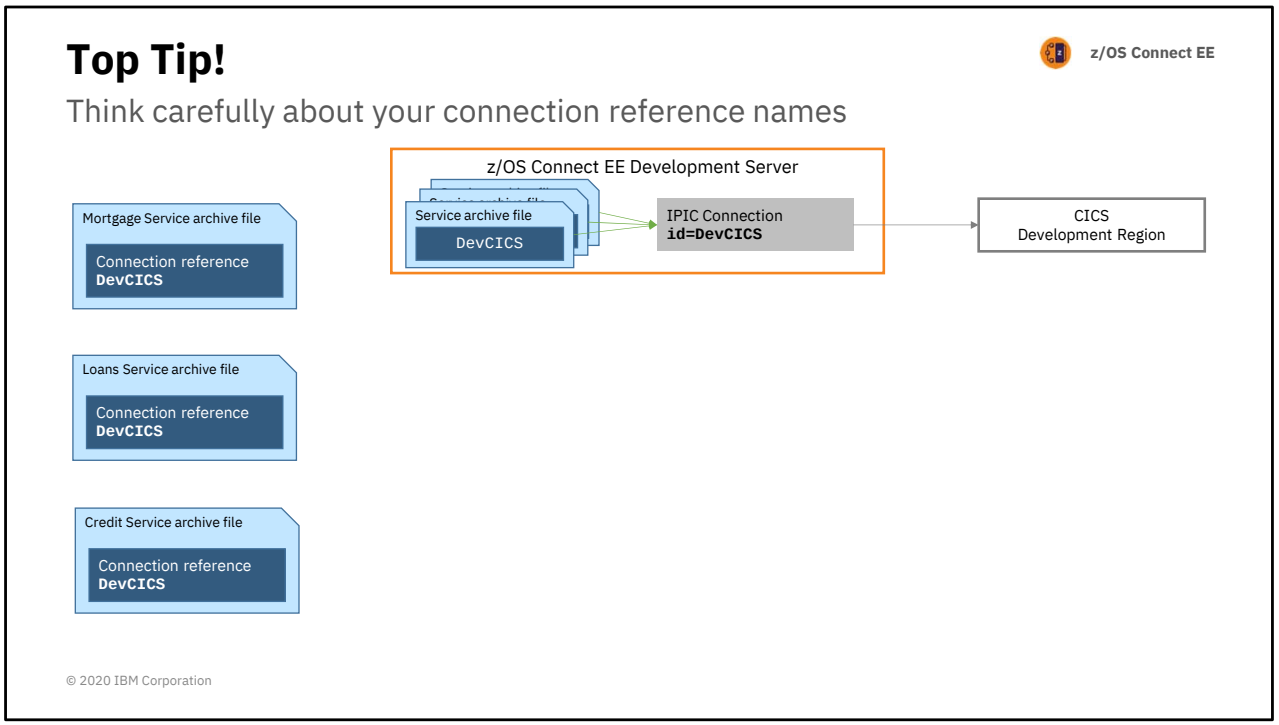
We use a POST request to create a new order. The JSON request message contains the item reference number and the number of items to be ordered. The JSON response message will contain a message that tells us whether the order has been successful.

Recommendations:

- Use consistent field names across different services
- Define enterprise API design standards and guidelines, including, use of HTTP verbs,

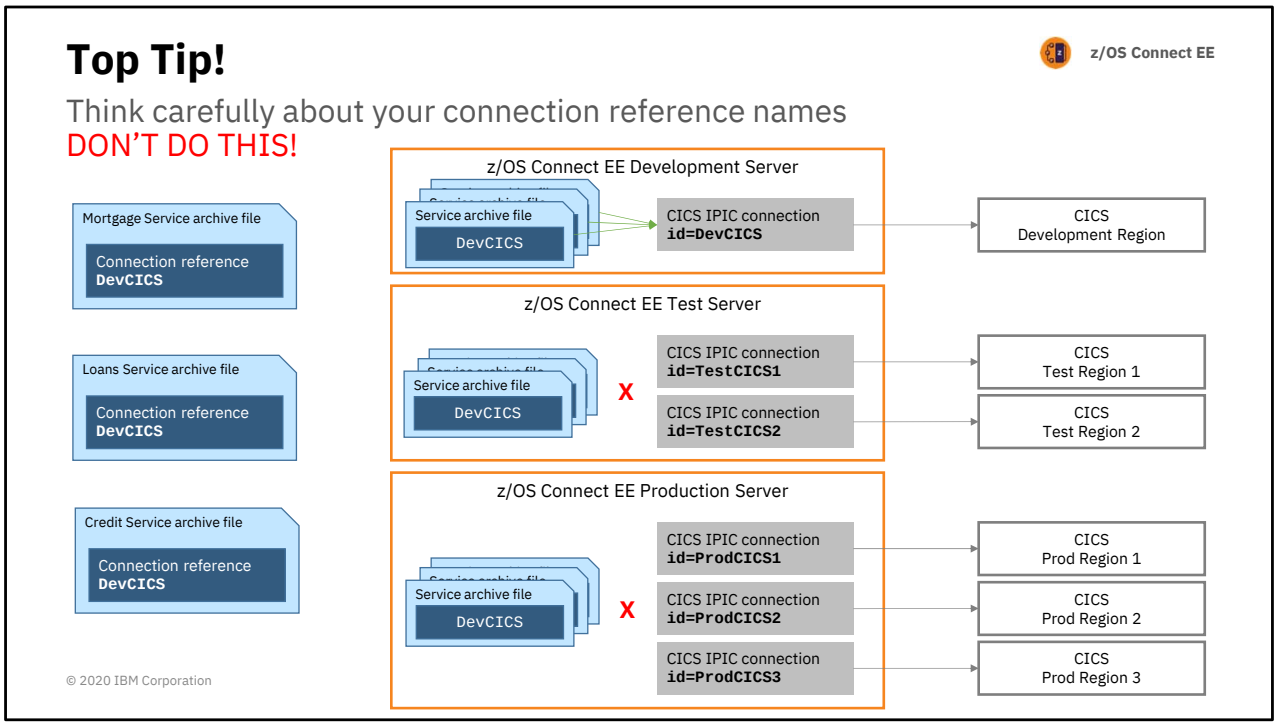
path parameters, query parameters, naming convention, use of plurals etc.

- Define guidelines for whether to do field assignment and omission in the service interface or the API. For example, if a service is used across multiple APIs, and field assignment and omission requirements are different for the APIs, it is recommended to do the field assignment and omissions in each API. However, if a field always needs to be assigned, or omitted, it is recommended to do the assignment or omission in the service interface.



As well as the JSON request and response mappings for the z/OS asset, a service archive contains a connection reference to the z/OS address space that hosts the program or data. When the service archive is deployed to a z/OS Connect EE server, this connection reference is associated with the id of a connection definition in the server.xml configuration file. In this example, it's an IPIC connection to a CICS development region. This is effectively an association between a logical name in the service archive and a physical connection definition in the server.

It's important that you use a logical connection reference name rather than a name based on the environment as shown in this example. The goal is to make these environment independent so that the same service archive can be deployed to dev, test and production z/OS Connect EE servers.

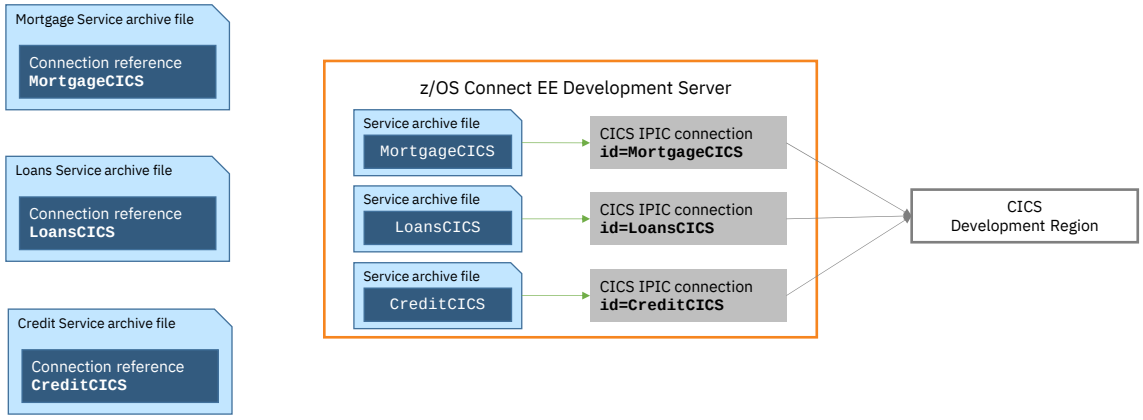


If you use a name like DevCICS for a connection reference name in the service archive, the problem comes when you want to deploy the same archive to a test or production server. You will not want to define a connection called DevCICS in the server.xml of a production server. So the service will not work.

Top Tip!

z/OS Connect EE

Think carefully about your connection reference names
DO THIS! Use logical rather than physical names



Instead you should use logical rather than physical connection reference names, for example;

- MortgageCICS for a reference to the system that hosts the Mortgage application
- LoansCICS to the system that hosts the Loans application
- CreditCICS to the system that hosts the Credit application

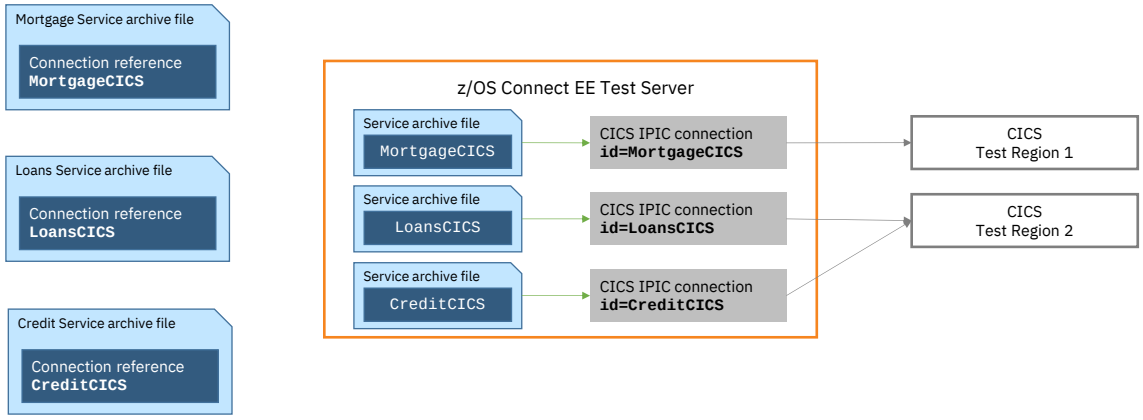
This allows you to deploy the service archives to different environments.

On this chart, the service archives are deployed to the development system where all 3 CICS applications run on the same CICS region. The 3 IPIC connections point to the same CICS development region."

Top Tip!

z/OS Connect EE

Think carefully about your connection reference names
DO THIS! Use logical rather than physical names

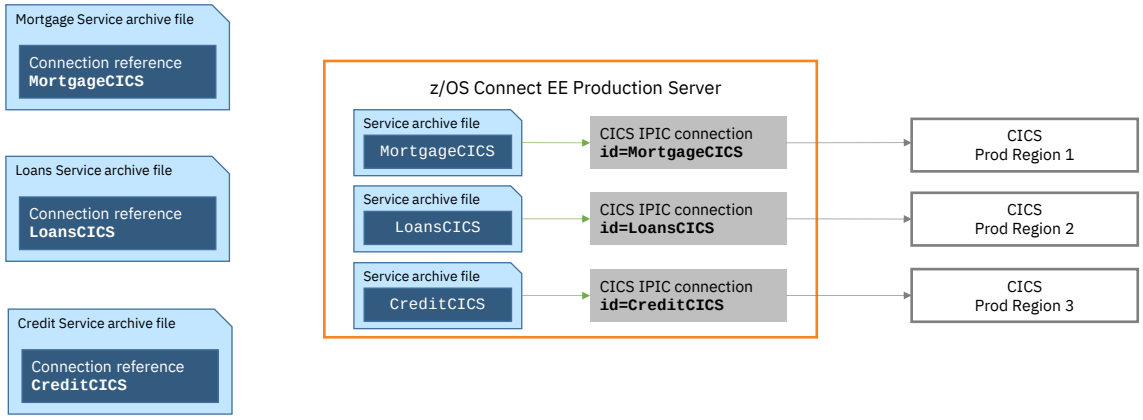


On this chart, the same service archives are deployed to the test system where the CICS Mortgage application runs in one test CICS region and the Loans and Credit applications both run in another region.

Top Tip!

z/OS Connect EE

Think carefully about your connection reference names
DO THIS! Use logical rather than physical names



On this chart, the same service archives are deployed to the production system where the 3 applications run in different CICS regions. In this case, the 3 IPIC connections point to different CICS production regions.

It is important to create a naming convention for connection references at the beginning of the project, because it’s difficult to change this later.

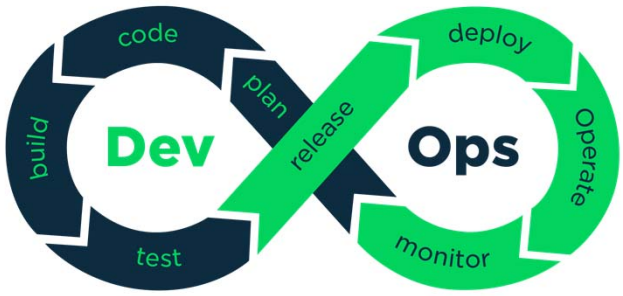
/devops

Building and Deploying z/OS Connect EE assets

What is DevOps?

Development + Operations

z/OS Connect EE



ORGANIZATIONS THAT HAVE IMPLEMENTED DEVOPS SAW THESE BENEFITS:

IMPROVED QUALITY OF SOFTWARE DEPLOYMENTS	63%
MORE FREQUENT SOFTWARE RELEASES	63%
IMPROVED VISIBILITY INTO IT PROCESS AND REQUIREMENTS	61%
CULTURAL CHANGE COLLABORATION / COOPERATION	55%
MORE RESPONSIVENESS TO BUSINESS NEEDS	55%
MORE AGILE DEVELOPMENT	51%
MORE AGILE CHANGE MANAGEMENT PROCESS	45%
IMPROVED QUALITY OF CODE	38%

Source: 2013 State of DevOps Report by PuppetLabs

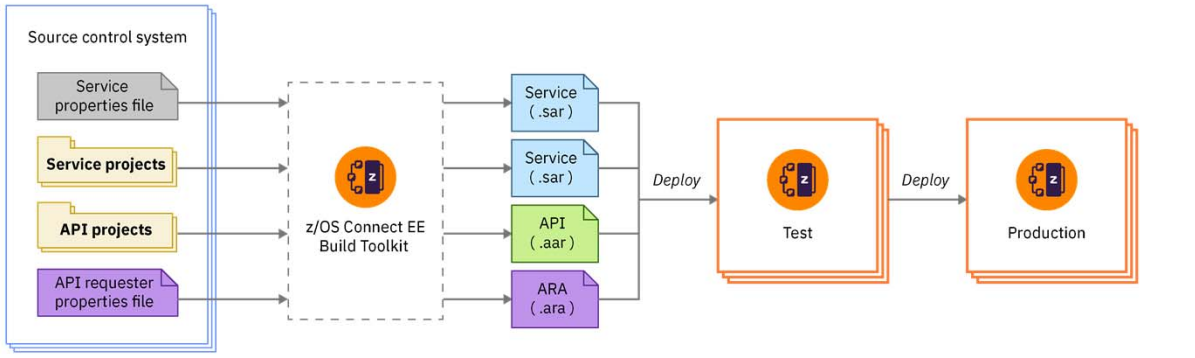
DevOps is a software development method that stresses collaboration between Software Developers (Dev) and Business Operations (Ops). It provides automation for Continuous Integration (CI) and Continuous Delivery (CD).

DevOps considerations

z/OS Connect EE

Automate the development and deployment of services, APIs, and API requesters for continuous integration and delivery.

- The build toolkit supports the generation of service archives and API archives from projects created in the z/OS Connect EE API toolkit
- The build toolkit also supports the use of properties files to generate API requester archives
- Run the build toolkit from a build script to generate the archive files
- Deploy of services, APIs and API requesters to z/OS Connect EE servers



ibm.biz/zosconnect-devops

Whereas the z/OS Connect EE API toolkit is used by developers to create and deploy APIs to development z/OS Connect EE servers, it is unlikely that you will use the API toolkit to deploy APIs to production servers. Instead, API toolkit project files will be stored in a Source Control System (like GitHub for example) and then the z/OS Connect EE build toolkit is used to generate the service, API and API requester archive files. The generated archive files are normally stored in an artifact repository, and an automated deployment script (using Jenkins for example) deploys the APIs or services to test and production servers using the z/OS Connect RESTful administration interface.

Below is a brief description of the different phases of a CI/CD pipeline:

Plan + code + test

- Think about which APIs to create, naming conventions, versioning (see next chart) and which Source Control Management (SCM) system to use
- Develop guidelines for using the API toolkit
- Decide on testing tools to use i.e Swagger UI, REST clients like Postman, Curl, httpie

Build

- Build .aar, .sar, .ara files using the build toolkit. Develop scripts using build pipeline tools

Deploy

- Deploy .aar, .sar, .ara files using the RESTful administration interface or Zowe CLI, or by copying the archives to the resources directories and refreshing the server – as part of a build pipeline (for example with Jenkins, UrbanCode Deploy, Maven)

Test

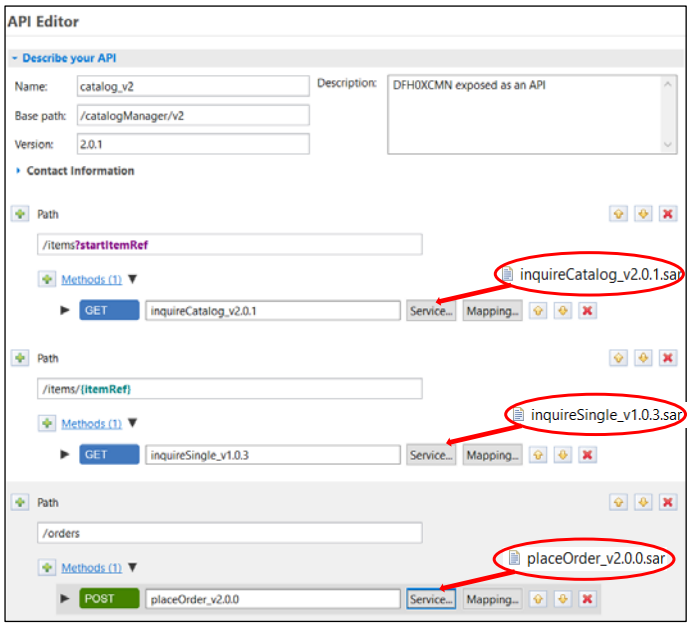
- Perform performance tests and enable API testing as part of the build pipeline

Release + operate + monitor

- Define guidelines for deployment across multiple z/OS Connect EE instances
- Define and document operational procedures
- Agree a monitoring strategy (for example using IBM OMEGAMON for JVM) and consider requirements for analytics and transaction tracking

API and service versioning example

z/OS Connect EE



Consider whether you need to assign a version to your APIs and services.

Assigning a new API version is necessary only if there are any breaking changes. For minor changes that do not break existing users, you can update the operations in place and use the version field in the API editor to track these minor version increments. For example, an operation can be added to an API or an optional field can be added to a schema without breaking the existing usage of an API.

Recommendations:

- Consider including a version identifier at the end of the service name (for example name-of-service_v2.1.0) to facilitate service versioning
- Consider including a version identifier at the end of the API name (for example name-of-API_v2.1.0)
- A version is usually described with 3 levels (M.m.p) where M=Major, m=minor and p=patch.
- A patch and a minor change should not change the interface and should therefore be backward compatible, however, to exploit the new functions added by a minor change, the REST client application may need to be updated. Therefore the version in the base

path does not need to include the minor and patch qualifiers.
- Include the version in the base path of the API (for example /catalogManager/v2) to facilitate API versioning. This method requires a separate API project for each version but has the advantage of producing separate Swagger documents, one for each version.
- Whereas a major change is expected to be disruptive and to modify the interface, thus the REST client application must be updated to call the new version of the API.

Note: A service version does not have to be related to an API version

For more information on API and service versioning see **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/designing/api_versioning.html**

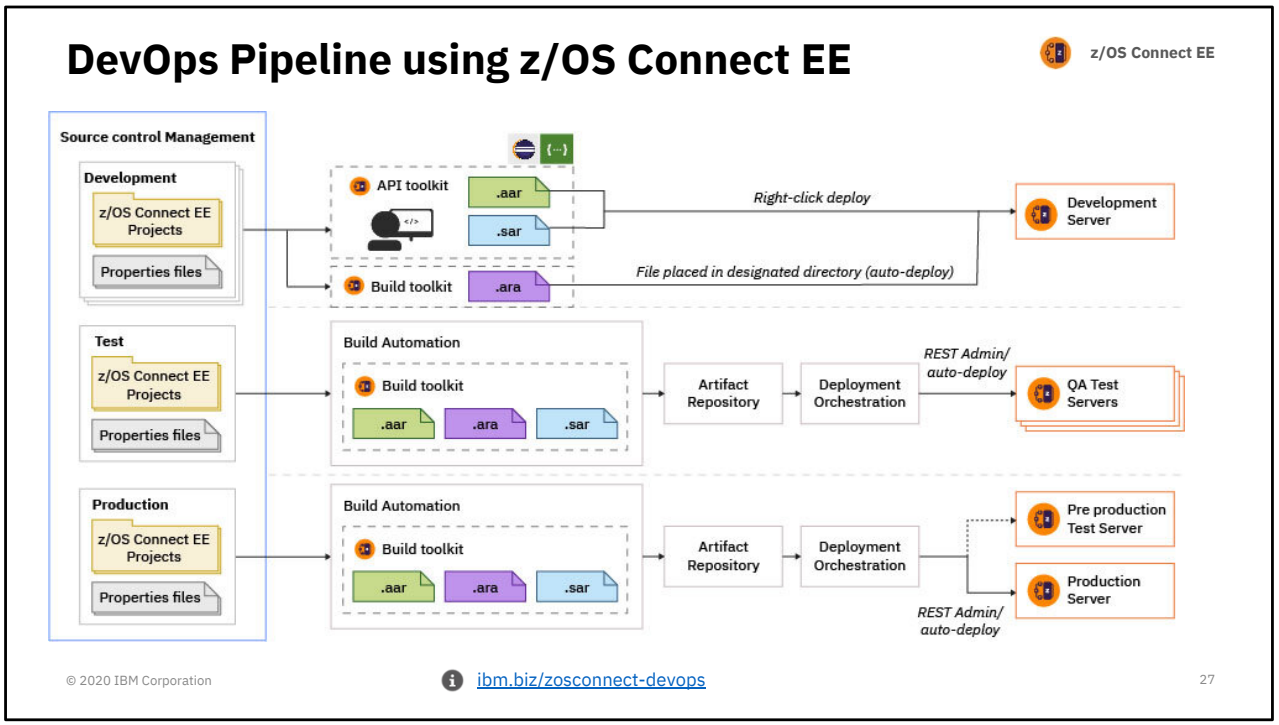
Important: Each API name, service name, and base path name must be unique within a server.

Top Tip!

An API versioning strategy for the enterprise will have been decided by the enterprise API management team in your organization

GO TALK TO THEM!

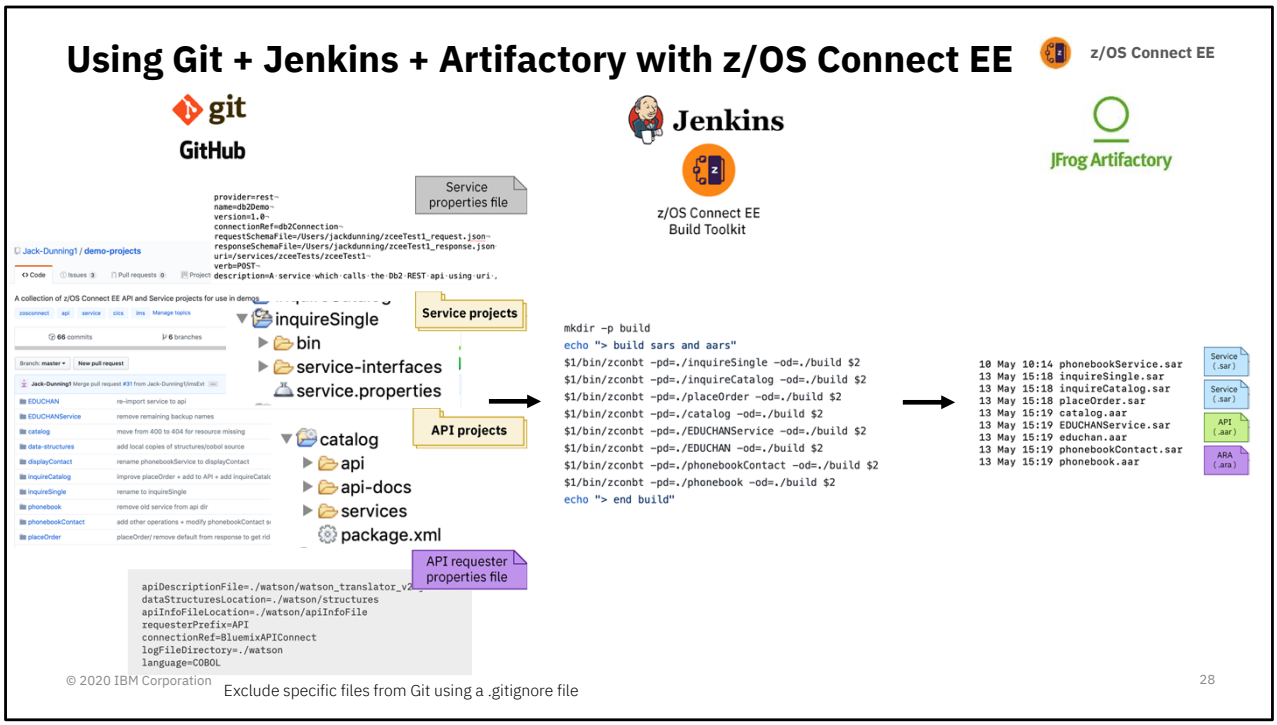
Consider z/OS Connect EE API and service versioning from the beginning. It's difficult to change it later



This chart shows an example DevOps pattern.

Recommendations:

- The API toolkit should be used to build and deploy .aars and .sars to development servers. This is intended as a developer tool to help them to quickly develop APIs.
- When development work is complete, the related project files should be stored in a Source Control Management (SCM) system.
- API and service projects and properties files should be treated as source code and managed by the SCM.
- Archive files (.aar, .sar and .ara) should **NOT** be stored in SCM but in an artifact repository
- The build toolkit should be used as part of build automation scripts to build .aars, .sars and .aras for test and production.
- Deployment Orchestration Automation is then used to deploy the stored archive files to test and production servers.
- Archive files are deployed either using the z/OS Connect EE RESTful administration interface or by copying the files to the designated directories for automated deployment



The developer commits the Eclipse API toolkit project in a Source Control Management (SCM) system like Git. And then a continuous integration tool like Jenkins clones the project, uses the z/OS Connect EE build toolkit to generate the deployable artefact (e.g an API archive) and stores it in an artefact repository like Jfrog Artifactory.

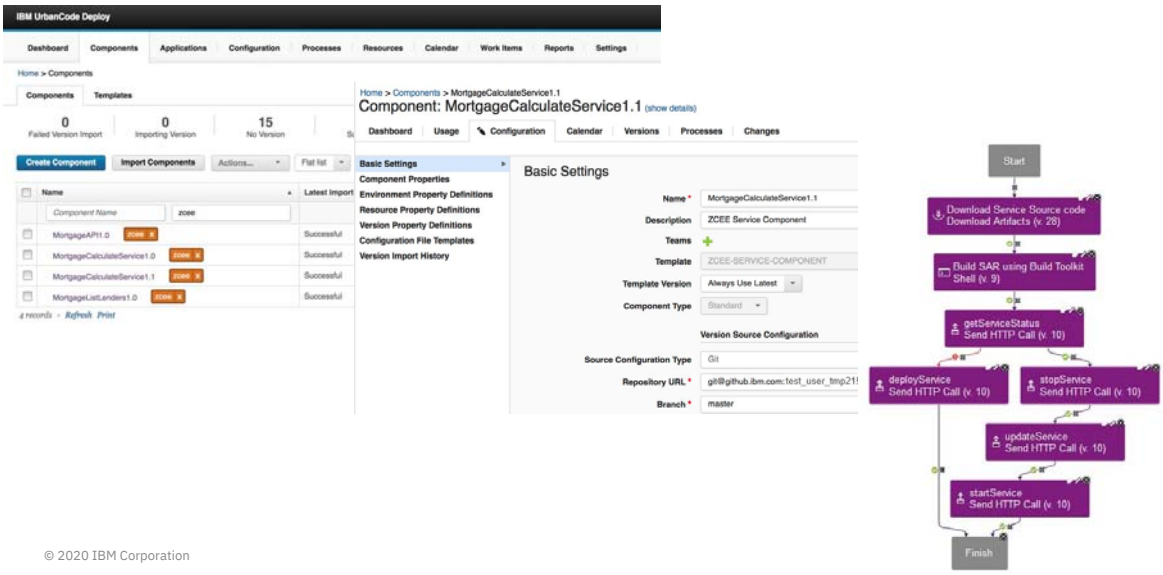
Note that the *bin directory in service projects should not be stored in the SCM.

Jenkins can be configured to monitor SCM repositories and automatically run builds when changes occur. This chart shows a simple script that uses the build toolkit to build sars and aars, which can then be stored in an artifact repository such as Artficatory, using further steps in the pipeline.

Using UrbanCode Deploy with z/OS Connect EE

z/OS Connect EE

Example implementation



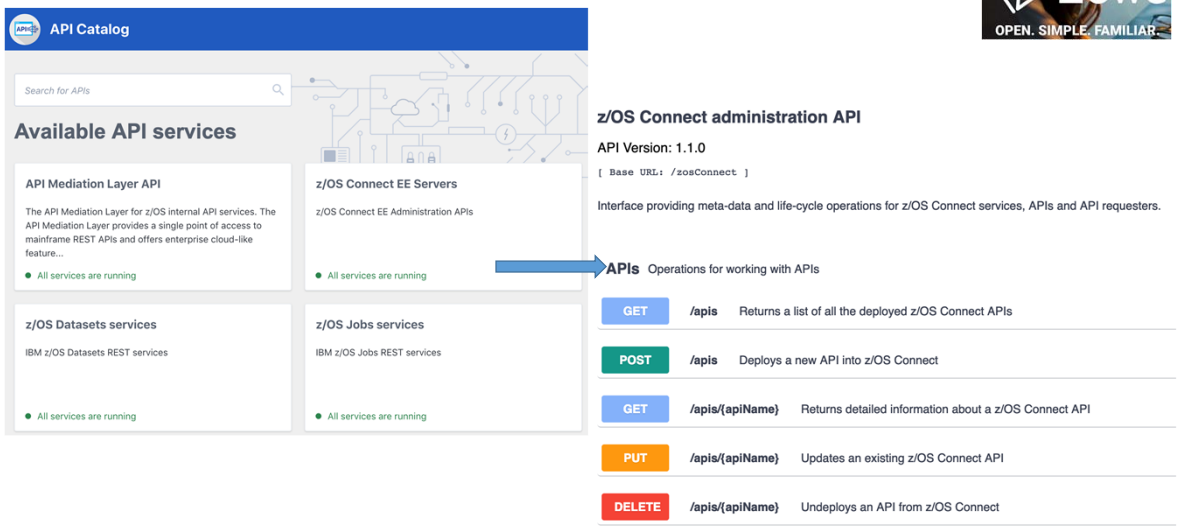
The z/OS Connect EE build toolkit can also be used with IBM UrbanCode Deploy (UCD). UCD can also group artefacts together and deploy them across different environments, for example, qualification, pre-production and production environments.

See the following blog for a sample implementation using Git repositories to store source code, and IBM UrbanCode Deploy (UCD) to run the z/OS Connect EE build toolkit to generate the archive files and deploy them:
https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/yun-han-li1/2020/08/12/sample-workflow-with-ibm-urbancode-deploy-and-git

Note: Other technologies are also available for building an API deployment pipeline.

z/OS Connect EE integration with Zowe

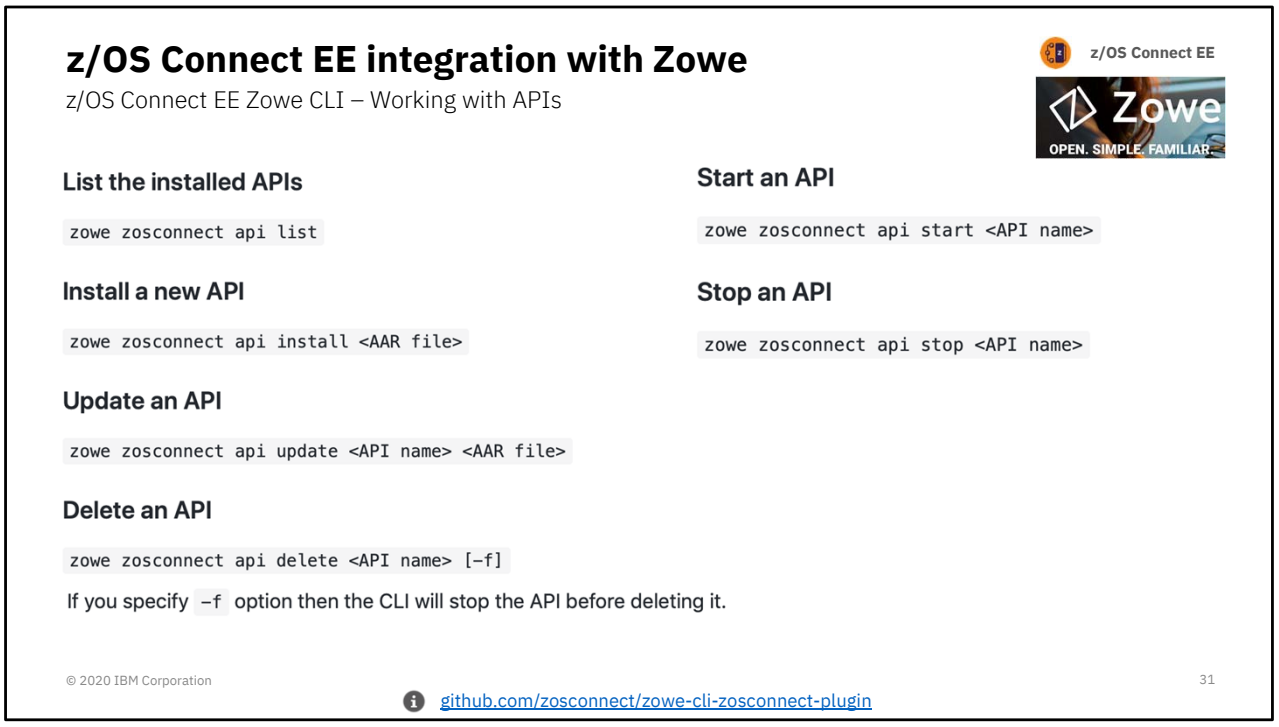
z/OS Connect EE and Zowe API Catalog – Administration API



A Zowe CLI can be used to script automated deployment and/or control of z/OS Connect EE resources such as APIs (AAR files), Services (SAR files) and API requesters (ARA files).

z/OS Connect EE also integrates with the Zowe API Mediation Layer which provides a single place where you can find all the APIs that are available on your mainframe and access them from a single well known HTTP endpoint. When you first install Zowe, you get the APIs for working with data sets, Jobs, z/OS MF and the API mediation layer itself. If you want to add your own APIs, such as the administration APIs for a z/OS Connect EE server, you can use Zowe to add an existing API without having to change anything in the server that provides the API. The blogpost link discusses how this is achieved using a sample configuration (see next slide).

See the following blog for more information on exposing z/OS Connect EE APIs in Zowe API Mediation Layer
https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/samantha-catling1/2020/08/11/expose-zos-connect-ee-apis-in-zowe-api-mediation-l



This slide shows the Zowe CLI syntax for working with APIs (aka AAR files). The GitHub link provides you with access to the Zowe CLI plugin that delivers the support for z/OS Connect EE.

This makes deployment even easier!

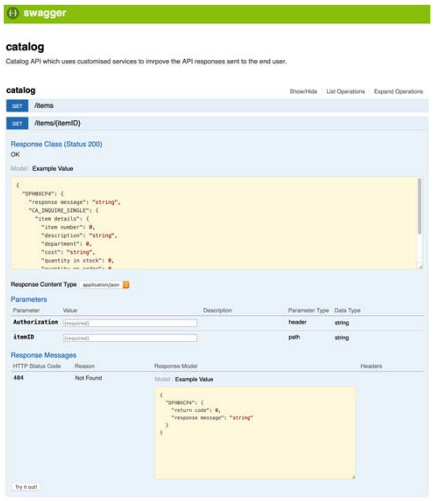
See the following blog for more information on using the Zowe CLI with z/OS Connect EE: **https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/samantha-catling1/2020/08/11/zowe-cli-plug-in-for-zos-connect-ee-v11**

Download the plugin here:
github.com/zosconnect/zowe-cli-zosconnect-plugin

Testing of APIs

Manual and automated testing

z/OS Connect EE



API Connect Test & Monitor

curl://

POSTMAN

httpie

urban{code} Deploy

This chart shows how Swagger UI can be used to test an API. It also shows some of the commonly used testing tools.

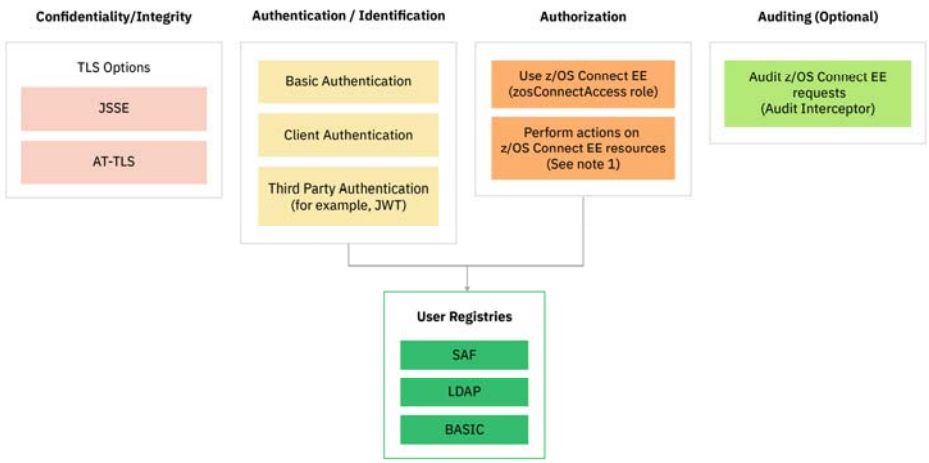
Note: other technologies are available for testing APIs.

/security

Securing APIs with z/OS Connect EE

High level security options available in z/OS Connect EE

z/OS Connect EE



http://ibm.biz/zosconnect-security

This chart shows the options provided by z/OS Connect EE for implementing the common security principals.

Confidentiality
Confidentiality ensures that an unauthorized party cannot obtain the meaning of the transferred or stored data. Typically confidentiality is achieved by encrypting the data.

Integrity
Integrity ensures that transmitted or stored information was not altered in an unauthorized or accidental manner. Typically it is a mechanism to verify that what is received over a network is the same as what was sent.

Authentication
Authentication is the process of validating the identity that is claimed by the accessing entity. Authentication is performed by verifying authentication information that is provided with the claimed identity. The authentication information is generally referred to as the accessor's credentials. A credential can be the accessor's name and password. It can also be a token provided by a trusted party, such as a JSON Web Token (JWT) or an X.509 certificate.
Authentication is usually one of the earliest steps in a request workflow. When

authenticated, an identity can be asserted to the downstream process steps, meaning that these steps trust that the identity was successfully authenticated by the upstream steps.

Identification
Identification is the ability to assign an identity to the entity attempting to access the system. Typically the identity is used to control access to resources. Depending on the security model in which the identification is performed, the identity may come from the authentication credentials or it might be asserted from another server.

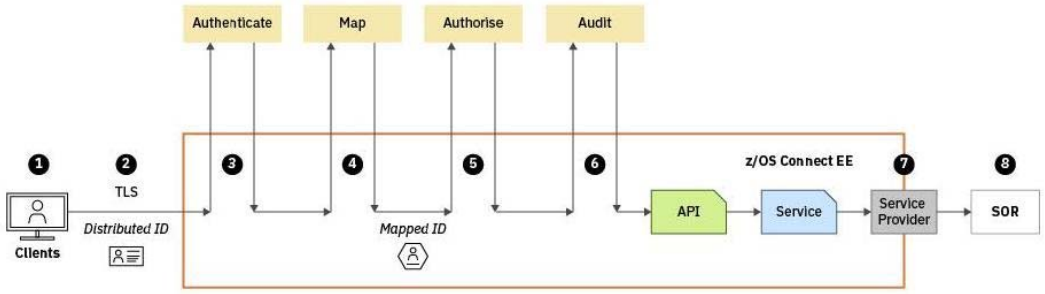
Authorization
Authorization is the process of checking whether an authenticated identity is to be given access to the resource that it is requesting. A typical implementation of authorization is to pass to the access control mechanism a security context that contains the authenticated identity.

Auditing
Auditing provides you with the ability to capture and record events such as an API request so that you can analyze them later, perhaps after a breach of your security has occurred.

We look at the options for enabling these security requirements in the following charts.

API provider security flow

z/OS Connect EE



1. Client credentials
2. Identity passed on connection
3. Authenticate the client
4. Map authenticated identity to a user ID
5. Authorize the authenticated user ID
6. Audit the request
7. Secure connection to System of Record
8. Use asserted identity in System of Record

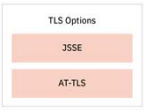
http://ibm.biz/zosconnect-security

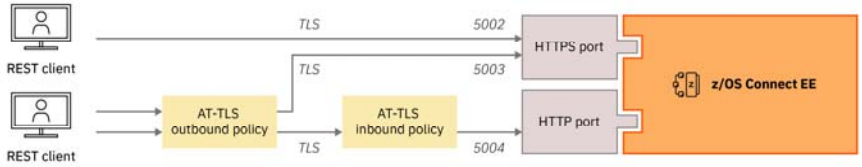
The API provider security flow includes the following security steps that can be performed by z/OS Connect EE

1. The credentials provided by the client. This can be a user ID and password, a third-party token or a TLS certificate.
2. The identity is passed on the connection between the client and the z/OS Connect EE server. This is typically a distributed ID, such as an X.500 distinguished name and associated LDAP realm, that originates from a remote system. Alternatively the identity could be a SAF user ID. The data sent on the connection can be encrypted using TLS.
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 z/OS Connect EE user registry.
5. Authorize the authenticated user ID to connect to z/OS Connect EE and to perform specific actions on z/OS Connect EE APIs or services.
6. Audit the API or service request.
7. Secure the connection to the System of Record (SoR) and (optinally) assert an identity to be used to invoke the program or transaction in the SoR.
8. The program or transaction runs in the SoR using the asserted identity.

Confidentiality / Integrity

z/OS Connect EE





http://ibm.biz/zosconnect-security

You can secure communications between a REST client and a z/OS Connect EE server by using the Transport Layer Security (TLS) protocol. TLS provides transport layer security that includes confidentiality, integrity, and authentication to secure the connection between a client and a z/OS Connect EE server.

z/OS Connect EE uses Java Secure Sockets Extension (JSSE) as the TLS implementation for secure connections. JSSE provides a framework and Java implementation that handles the handshake negotiation and protection capabilities that are provided by TLS.

Alternatively you can use Application Transparent Transport Layer Security (AT-TLS), a capability of z/OS Communications Server, for transport layer security with z/OS Connect EE.

Note: z/OS Connect EE is an unaware AT-TLS application and therefore does not have access to the partner certificate. This means that a z/OS subsystem cannot use a client certificate to authenticate with z/OS Connect EE when the connection between the z/OS subsystem and z/OS Connect EE is secured using AT-TLS.

Recommendations:

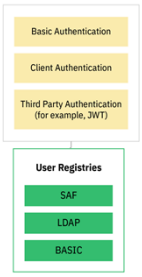
- Specify a Common Name (CN) for the z/OS Connect EE server certificate that is the same

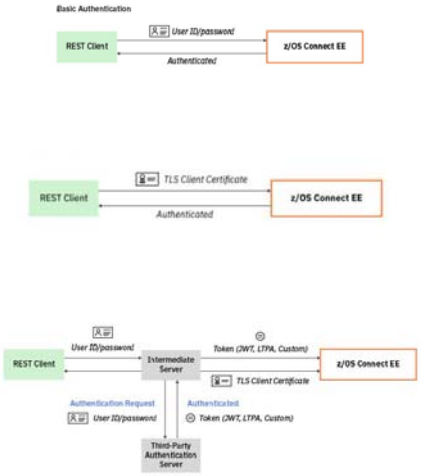
as the hostname that will be used by the REST API clients.

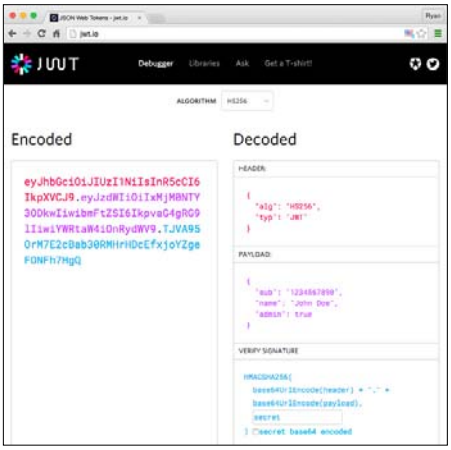
- Share key rings and certificates across a set of cloned z/OS Connect EE instances.
- The latest version of TLS (TLS v1.2) provides for the most secure range of ciphers.
- Control what ciphers can be used in the z/OS Connect EE server rather than in the REST client.
- Asymmetric encryption is much more expensive than symmetric encryption due to the large key sizes required, so it is important to minimize the number of TLS handshakes by persisting connections.

Authentication / Identification

z/OS Connect EE







http://ibm.biz/zosconnect-security

Three methods are provided for authentication between clients and a z/OS Connect EE server:

- Basic authentication (**Recommendation**: Use https so that the user's password is encrypted)
- Client certificate authentication
- Third-party authentication (most commonly used). **Recommendation**: Use authentication filters to enable different authentication options to be used depending on the request

Third party options:

- JSON Web Token (JWT) is the most popular third-party authentication token used with z/OS Connect EE
- OAuth 2.0 access token - the OAuth 2.0 protocol facilitates the authorization of one site to access and use information that is related to the user's account on another site.
- Security Assertion Markup Language (SAML) token
- Lightweight Third-Party Authentication (LTPA) token

A JWT consists of a:

1. Header – algorithm, token type
2. Payload – claims (issuer, subject, audience, expiry)
3. Signature – for integrity

For more information on authenticating with a JWT see
https://www.ibm.com/support/knowledgecenter/en/SS4SVW_3.0.0/securing/provider_third_party_auth.html#provider_third_party_auth__section_authenticate_jwt

Top Tip!

A lot of the security requirements will have been decided by the enterprise security architects in your organization

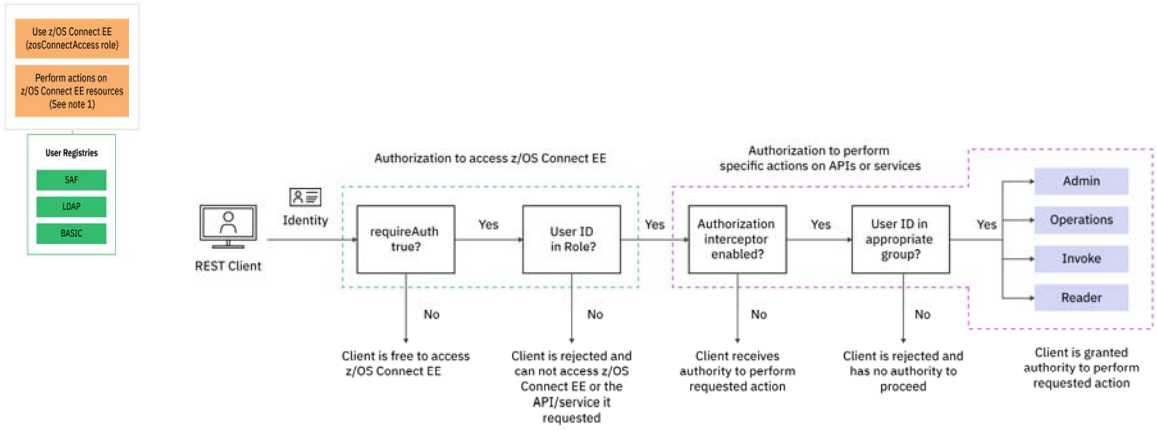
GO TALK TO THEM!

You'll then know which options you need to implement!

JSON Web Token (JWT) is the most popular third-party authentication token used with z/OS Connect EE

Authorization

z/OS Connect EE



http://ibm.biz/zosconnect-security

The z/OS Connect EE roles allow for separation of responsibilities for different users.

Admin
All z/OS Connect EE actions are allowed, including all *Operations*, *Invoke*, and *Reader* actions.

Operations
Ability to perform all z/OS Connect EE operations and actions except for *Invoke*.
The following actions are allowed:
- Obtain a list of APIs. For an individual API, get details and API Swagger document, deploy, update, start, stop, and delete.
- Obtain a list of services. For an individual service, get details, request and response schemas, statistics, deploy, update, start, stop, and delete.

Invoke
Ability to invoke APIs and services. *Invoke* authority does not provide access to z/OS Connect EE *Operations* actions.

Reader
Ability to: Obtain a list of APIs. For an individual API, get details and API Swagger document.

Obtain a list of services. For an individual service, get details, request and response schemas.

Note: Authorization requires authentication first to be successful.

Recommendations:
Configure interceptors at the global level so that APIs do not need to be individually defined in server.xml.

Auditing

z/OS Connect EE

- SMF123_SERVER_SECT_VERSION: 1 *Server Section*
- SMF123_RESERVED_02: 000000
- SMF123_SERVER_SYSTEM: ZT01
- SMF123_SERVER_SYSPLEX: ZT00PLEX
- SMF123_SERVER_JOBID: STC07299
- SMF123_SERVER_JOBNAME: MOPZCEP
- SMF123_SERVER_STOKEN: 5686536700910
- SMF123_SERVER_CONFIG_DIR: /var/zosconnect/servers/MOPZCEP/
- SMF123_SERVER_VERSION: 3.0.30.0

- SMF123_REQ_DATA_VERSION: 1
- SMF123S1_REQ_TYPE: 1
- SMF123S1_HTTP_RESP_CODE: 200
- SMF123S1_RESP_FLAGS: 0
- SMF123S1_RESERVED_04: 000000
- SMF123S1_USER_NAME: NicolasBoss
- SMF123S1_USER_NAME_MAPPED: ZCOBOSS
- SMF123S1_CLIENT_IP_ADDR: 9.101.139.230
- SMF123S1_API_NAME: catalog_v1.0
- SMF123S1_API_VERSION: 1.0.0
- SMF123S1_SERVICE_NAME: inquireCatalog_v1.0
- SMF123S1_SERVICE_VERSION: 1.0.0
- SMF123S1_REQ_METHOD: GET
- SMF123S1_REQ_QUERY_STR: startItemRef=0
- SMF123S1_REQ_TARGET_URI: /catalogManager/v1.0/items
- SMF123S1_REQ_PAYLOAD_LEN: 0
- SMF123S1_RESP_PAYLOAD_LEN: 1923
- SMF123S1_TIME_ZC_ENTRY: 2020-03-09|13:29:53.105249
- SMF123S1_TIME_ZC_EXIT: 2020-03-09|13:29:53.162643
- SMF123S1_TIME_SOR_SENT: 2020-03-09|13:29:53.142812
- SMF123S1_TIME_SOR_RECV: 2020-03-09|13:29:53.144512
- SMF123S1_SP_NAME: CICS-1.0
- SMF123S1_SOR_REFERENCE: CICSMOB1
- SMF123S1_SOR_IDENTIFIER: MOPZT00.CICSMOBP
- SMF123S1_SOR_RESOURCE: MZIC,DFH0XCMN
- SMF123S1_REQ_ID: 13
- SMF123S1_TRACKING_TOKEN: C2C1D80100…
- SMF123S1_REQ_HDR1: User-Agent:PostmanRuntime/7.22.0
- SMF123S1_REQ_HDR2: Host:9.212.143.123:50743
- SMF123S1_REQ_HDR3:
- SMF123S1_REQ_HDR4:
- SMF123S1_RESP_HDR1:
- SMF123S1_RESP_HDR2:
- SMF123S1_RESP_HDR3:
- SMF123S1_RESP_HDR4:

CICS subsystem *Request Data Section*

Auditing allows for accountability for who has done something.

This chart illustrates the SMF record information written by z/OS Connect EE for each API request.

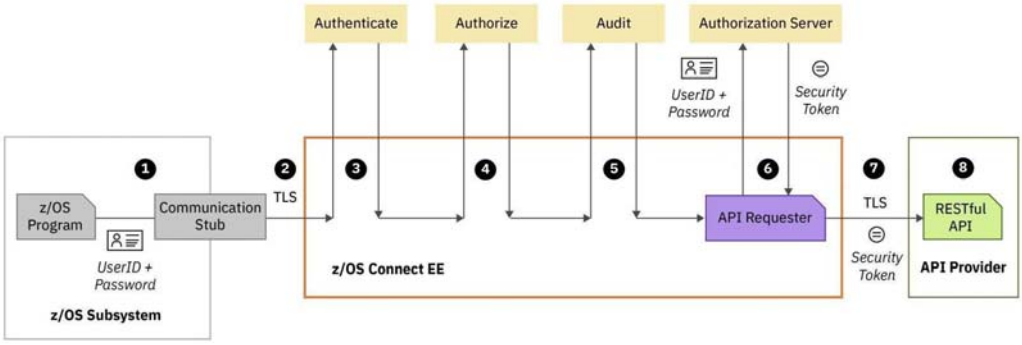
Note: For version 2, records are written every 20 requests.

For more information on the structure of the message, see **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/configuring/auditing_intro.html**

You can also use the IBM Common Data Provider for z Systems to stream z/OS Connect EE audit records to an analytics platform like Splunk. See the Monitoring section of this presentation.

API requester security flow

z/OS Connect EE



1. z/OS program can provide user ID & password
2. Send request on secure connection
3. Authenticate the credentials
4. Authorize the authenticated user ID
5. Audit the request
6. Obtain token from authorization server
7. Secure connection to API provider with security token
8. RESTful API runs in API provider

http://ibm.biz/zosconnect-security

The security principles also apply to API requester.

The API requester flow includes the following security steps that can be performed by z/OS Connect EE

1. A user ID and password can be provided by the CICS, IMS or z/OS application.
   - The user ID and password can be used for basic authentication by the z/OS Connect EE server.
   - The user ID and password can also be used to obtain a token from an authorization server to use on the request to the RESTful API.
2. The connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server. The data sent on the connection can be encrypted using 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. The RESTful API runs in the external API provider.

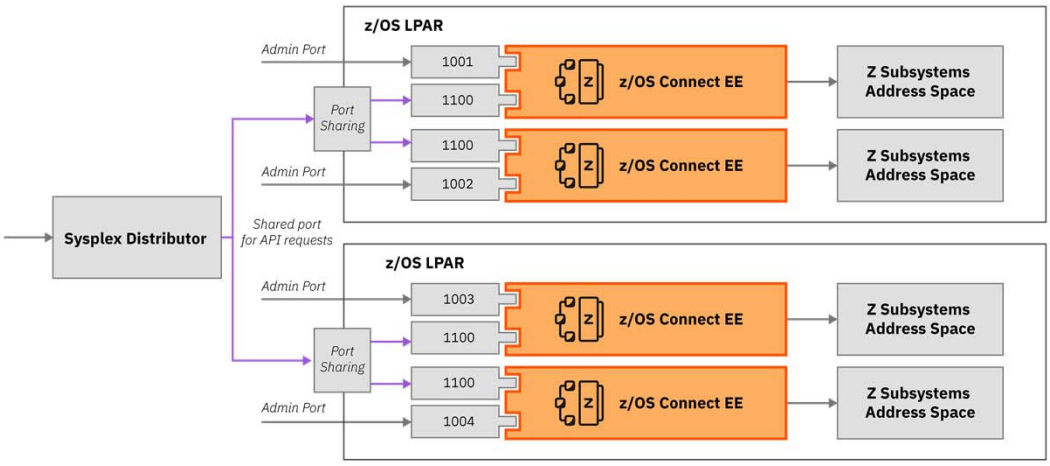
/workload management

High availability

High Availability

Topology

z/OS Connect EE



ibm.biz/zosconnect-ha-concepts

ibm.biz/zosconnect-scenarios

An HA z/OS Connect EE environment can use TCP/IP load balancing technologies to distribute connections from clients across multiple servers.

TCP/IP port sharing enables a group of cloned z/OS Connect EE servers running on the same LPAR to listen on the same port. The shared point is defined in the TCP/IP profile as below:
1100 TCP ZCTACX* SHAREPORT ; Z/OS CONNECT EE HTTPS PORT

It is possible to combine the use of Sysplex Distributor with TCP/IP port sharing for a high availability configuration. Then the Sysplex Distributor distributes requests across LPARS, and port sharing distributes requests across different subsystems within an LPAR.

New connections are distributed across the servers using a weighted round-robin algorithm based on the efficiency of the server application in accepting new connection requests and managing the socket backlog queue. Alternatively, the SHAREPORTWLM option can be used so that the server selection is based on WLM server-specific recommendations.

Recommendation: The cloned servers should share the same TCP/IP port for API invocation requests, however, each server will need a specific admin port if the RESTful administration interface is to be used so that admin (e.g deploy API) requests can be targeted at specific

servers.

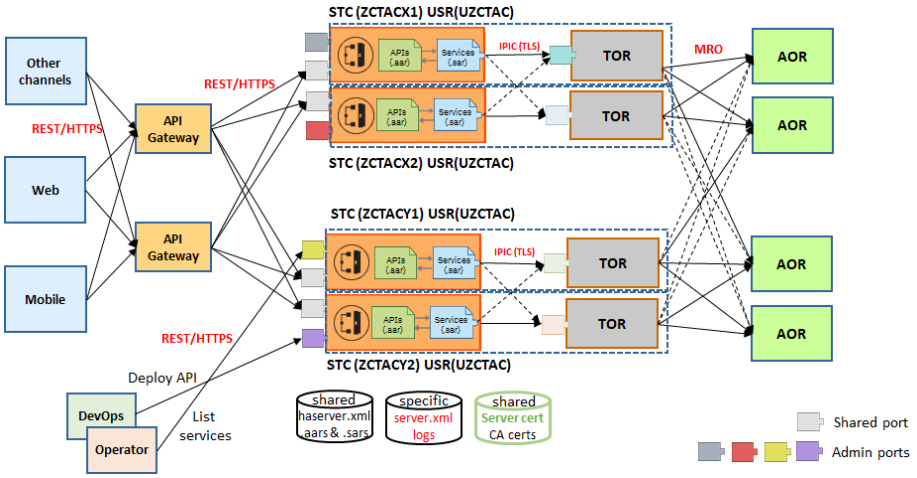
It is also necessary to plan for variations in workloads, especially increased workloads, when it might be necessary to increase the capacity of the system. The system must respond in a predictable way to workload variation to meet SLAs. For z/OS Connect EE, scalability can be achieved by manually increasing the number of servers.

For a more detailed high availability scenario, see
ibm.biz/zosconnect-scenarios

Example high availability configuration

z/OS Connect EE

- Use TCP/IP load balancing technologies to distribute connections from clients across multiple servers
- Naming convention for z/OS Connect EE servers, proc names , user IDs etc. is important
- Use different ports for API invocations and RESTful administration interface requests
- Share resources like config files, APIs and services, and SAF keyrings and certificates across servers



Recommendation: Use a server naming convention that allows you to define a shared port. Using this configuration, a workload of API requests will be balanced across the z/OS Connect EE servers ZCTACX1 and ZCTACX2 .

Recommendation: For planned outages of a z/OS Connect EE server that is running as part of an HA environment, consider using the following MVS command to pause the HTTP port being used by the server:
MODIFY <jobname>.<identifier>,PAUSE,TARGET='httpendpoint'

For IP-based service providers (CICS, IMS, REST) an API request received by a z/OS Connect EE instance in LPAR1 can be sent to a backend system on the same LPAR or another LPAR.

Recommendation: Consider using the Sysplex distributor OPTLOCAL keyword (on VIPADISTRIBUTE statement) which sets preference to local (in-LPAR) connections over remote connections. This will reduce cross-LPAR communication.

When you have many cloned z/OS Connect EE servers configured to manage workload, you can share resources across the clones, including:

- Configuration files
- API, service and API requester directories

- TCP/IP ports
- STC user IDs
- RACF keyring and certificates

Recommendation: If a set of cloned servers are sharing resources then, the RESTful administration interface should be used to deploy the API to only one of those servers and then the Modify refresh command should be issued to the other cloned servers.

For information on how to share files across different z/OS Connect EE server instances see **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/highavailability/share_server_config.html**

Top Tip!

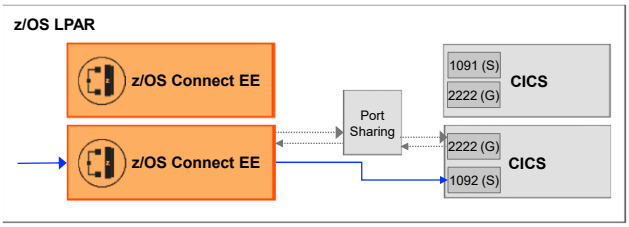
Create an enterprise z/OS Connect EE Installation Standards document

UPDATE THE DOCUMENT REGULARLY

And ensure that different teams follow the same standards

IPIC HA

z/OS Connect EE



Enquire on generic port (G) to identify specific port (S)

Establish connection to CICS specific port (S) and flow requests

- The CICS Service Provider supports high availability IPIC connections
- CICS regions listen on two end points
  - Generic port shared by all regions in the cluster (connections can be balanced)
  - Specific port used exclusively by a specific region
- Specify **`sharedPort=true`** on **`zosconnect_cicsIpicConnection`** in `server.xml`

When using IPIC HA, a z/OS Connect EE server connects to a CICS region in the cluster by using the generic end-point. The connection request to the generic end point is intercepted by the connection-balancing mechanism and is routed to a generic TCPIPSERVICE that belongs to one of the CICS regions in the cluster. The selected CICS region then returns the IP address and port of its specific end point to the z/OS Connect EE server, and the connection is established to the specific end point.

The IPCONN may be auto-installed, or pre-defined.
Recommendation: When using security, ensure that the generic and specific TCPIPServices specify the same security credentials.

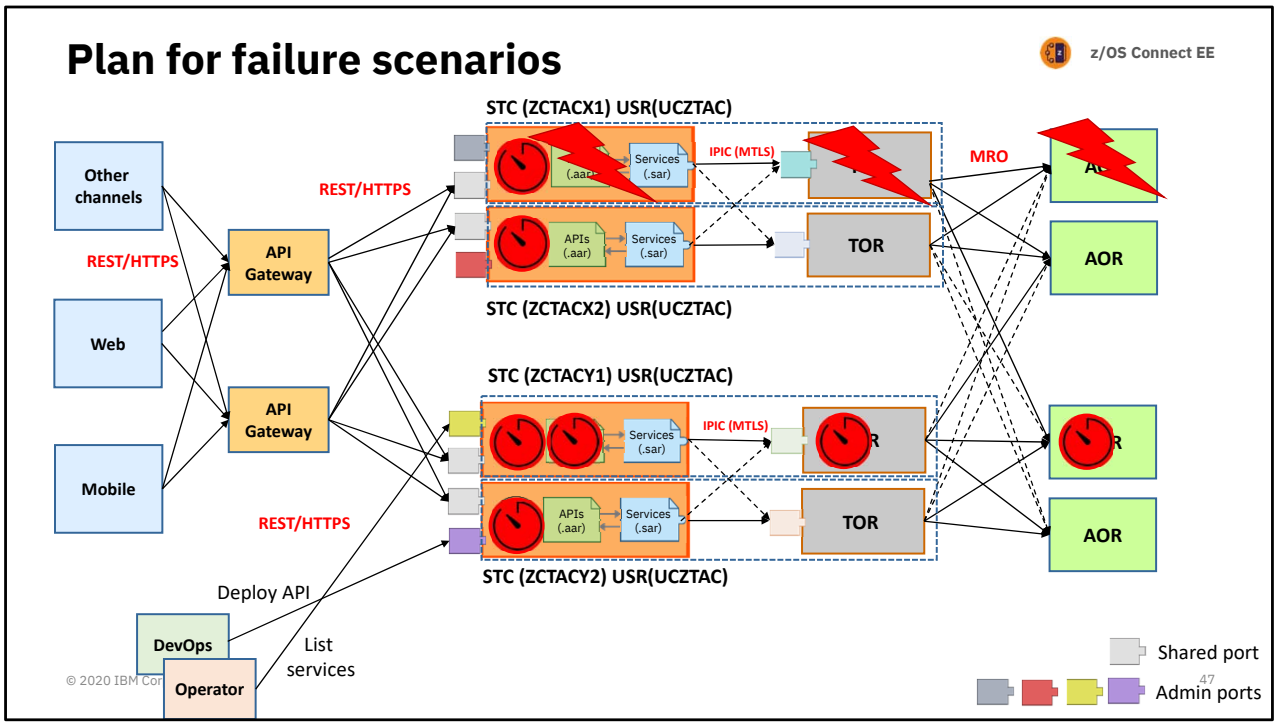
z/OS Connect EE sends a heartbeat request every 30 seconds. If it has not heard from CICS in the last 30 seconds, and if it does not get a response from a heartbeat request, it closes the connection. If this is an IPIC HA connection then normally the connection will get re-established to a different CICS. However, if it tries to re-establish the connection back to the same CICS and a pre-defined IPCONN resource definition has been configured, this can fail if the IPCONN has not been released.

Recommendation: If predefined IPCONNs are used, do one of the following:
- If the IPIC connection does not have **sharedPort=true**, disable the heartbeat by

setting **heartbeatInterval=0** on the **cicsIpicConnection**

- Create your own version of mirror transaction (CSMI) with **SPurge** set to YES, and specify this in the service configuration or in the **transid** attribute on the **cicsIpicConnection**

Note: If an IPIC HA connection to CICS1 fails, it is re-established to a different CICS (CICS2). However, when CICS1 is restarted, it will not receive requests until IPIC HA reestablishes a connection from zCEE to CICS1, either because the connection to CICS2 becomes unresponsive and is closed or drops, or it is manually released from the CICS side



Like with any new infrastructure it is necessary to plan for failures and scheduled outages. This chart shows the type of failures to consider, and below we give some general recommendations for different failure scenarios.

Recommendations:

z/OS Connect EE server failure

- Ensure z/OS Connect EE connections do not persist indefinitely so that a restarted z/OS Connect EE server receives new requests. Set **maxKeepAliveRequests** on the **httpOptions** element.
- For *planned* outages, pause HTTP ports before shutting down the z/OS Connect EE server

CICS AOR failure

- Use dynamic routing of DPL requests from TORs to AORs

CICS TOR failure

- Enable IPIC HA

- Note that a manual procedure is required in order to reconnect z/OS Connect EE instance 1 to TOR 1 after the TOR is restarted

z/OS Connect EE server slowdown

- Monitor API response times (for example using OMEGAMON for JVM)
- Monitor JVM metrics (for example using OMEGAMON for JVM or the IBM Health Center)
  - Heap size (current and max)
  - Number of GCs
- Consider option to use WLM based algorithms for Sysplex Distributor and TCP/IP port sharing – connection requests can be routed based on whether servers are meeting their WLM performance goals

CICS AOR slowdown

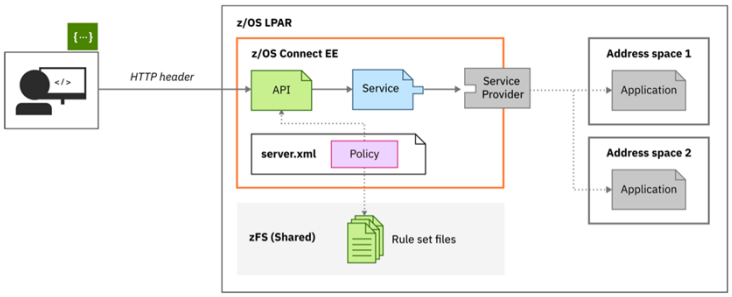
- Consider use of **DTIMOUT** to timeout deadlocked transactions in AORs
- Specify appropriate timeout period to timeout zCEE requests by setting **asyncRequestTimeout** (default timeout is 30 seconds)
- Ensure that CICS max tasks is set sufficiently high and consider use of TRANCLASS for TOR transactions to protect against specific poorly performing service
- Monitor API response times

CICS TOR slowdown

- Specify **asyncRequestTimeout** to timeout API requests
- Consider use of **TRANCLASS** for TOR transactions to protect against specific poorly performing service
- Monitor API response times

Policy-based API processing

z/OS Connect EE



Currently supports CICS, IMS and Db2 services.

Policies can be configured globally for every API in the server or for individual APIs

Policy-based API processing gives you an effective way to manage client requests that need to drive different behaviours in the backend program depending on variations in an HTTP header of the API request.

This gives you the ability, for example, to handle application affinities, where API requests need routing to specific regions.

ibm.biz/zosconnect-policy-intro

You create rule sets to define the condition and actions, then enable z/OS Connect EE policies to apply those actions to API requests. A rule set contains one or more rules that define the condition and actions.

Some example use cases are:

- Determine which CICS, IMS or DB2 system to connect to based on the value of the HTTP header
- Determine which CICS or IMS transaction code to use DB2 collection name based on the value of the HTTP header

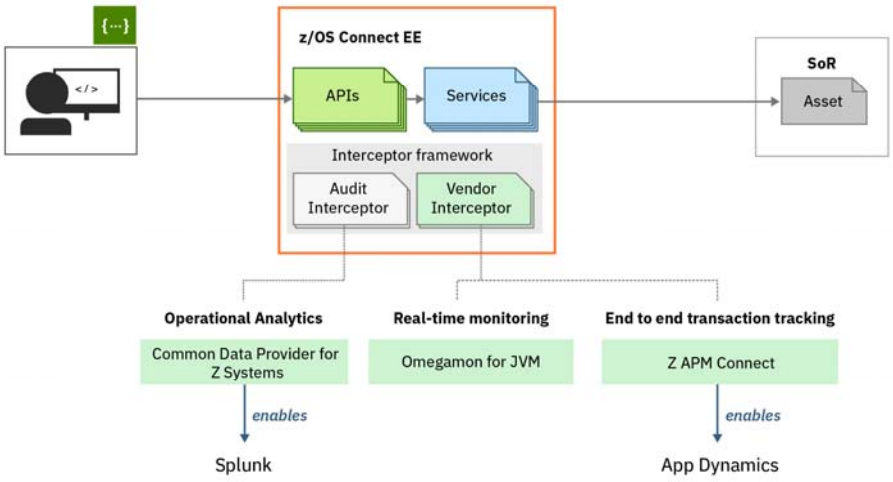
Learn more about this feature here:
ibm.biz/zosconnect-policy-intro

/monitoring

Real time monitoring, transaction tracking and operational analytics

Extending z/OS Connect EE for monitoring

z/OS Connect EE



ibm.biz/zapi-monitoring

As you prepare to deploy APIs to z/OS Connect EE, you might be considering how to monitor API workloads, track API requests across the enterprise and perform operational analytics. This chart shows how the interceptor framework of z/OS Connect EE can be used with monitoring solutions to support all these requirements.

Monitoring APIs with OMEGAMON for JVM

z/OS Connect EE

Monitoring APIs and services

Monitoring connections to SoRs

Monitoring SoR resources

ibm.biz/zapi-monitoring

How do you monitor a z/OS Connect EE API workload?

IBM OMEGAMON for JVM V5.4.0 supports z/OS Connect EE request monitoring by providing the System of Record (SoR) element of total response time, as well as API response time metrics.

The 'top level' OMEGAMON for JVM views of the **Enhanced 3270 UI** allow operations staff to see a quick health status of z/OS Connect EE. This chart shows the following views:

1. Monitor the APIs and services - this screen shows a top-level view of the APIs deployed to a z/OS Connect EE server. The screen shows the catalog and phonebook APIs. It shows request counts for the different HTTP methods used to invoke each API, for example, the catalog API is invoked using the GET and POST methods. It also shows that there are no errors.

2. Connections in server.xml – this screen shows the request counts for the different connections defined in server.xml

3. Monitor SoR resources – this screen shows more information on the SoR resources, for example, for the CICS service provider the SoR resource consists of the CICS trans ID and program name and for the IMS service provider, the SoR resource is the IMS trans ID

Recommendations:

- Use OMEGAMON for JVM to identify response time issues and to monitor the JVM heap size and garbage collection frequency
- Configure OMEGAMON for JVM to create alerts when abnormal events occur e.g API response time greater than 1 second.
- Integrate OMEGAMON for JVM with Systems Automation so that situations can be identified quickly and automated corrective actions can be taken.

End-to-end transaction tracking

z/OS Connect EE

ibm.biz/zapi-monitoring

How do you track requests from a REST client application to z/OS Connect EE, and all the way through to the System of Record (SoR)?

AppDynamics is an Application Performance Monitoring (APM) tool that automatically discovers, maps, and visualizes business transactions as they are processed across multiple servers. This chart shows how you can use AppDynamics to track API requests processed by z/OS Connect EE. The tracking information that is being used in this example is coming from Z APM Connect.

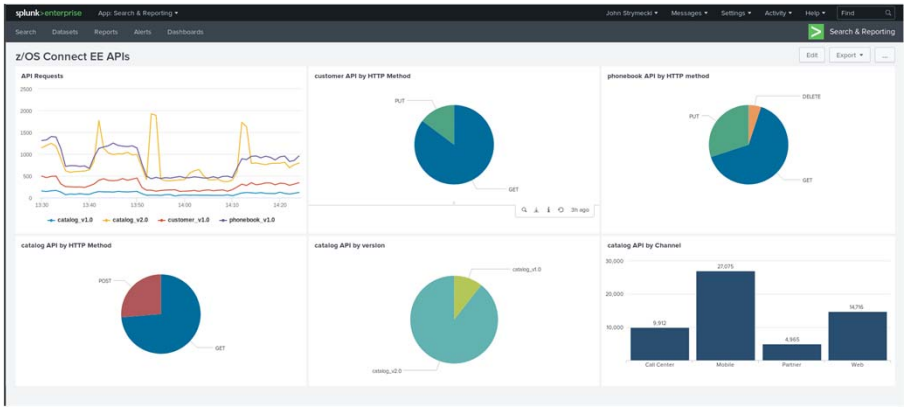
In this example, we see the Online banking application making API requests to a z/OS Connect EE server. We also see that the CICS Service Provider is being used to send requests to 4 CICS regions. We can monitor the number of requests and response times.

The tracking information that is being used in this example is coming from IBM Z Application Performance Management Connect (Z APM Connect). Z APM Connect provides transaction tracking information and resource monitoring metrics from a wide range of z/OS subsystems (including z/OS Connect EE) to APM solutions including AppDynamics and IBM Application Performance Management.

Operational analytics

z/OS Connect EE

Audit z/OS Connect EE requests (Audit Interceptor)



JOBNAME	ARRIVAL-DATE	ARRIVAL-TIME	TARGET-URI	SERVICE-OR-API	METHOD	USER-NAME	MAPPED-USER-NAME
MOPZCEW	2019/05/14	12:44:31.78	/catalogManager/v1.0/orders	catalog_v1.0	GET	JeanLeclerc	EMPLOY1

ibm.biz/zapi-monitoring

How do you collect operational data from z/OS Connect EE and stream it, in a consumable format, to an analytics platform such as Elasticsearch, Apache Hadoop or Splunk?

The IBM Common Data Provider for z Systems provides the infrastructure for accessing IT operational data from z/OS systems and streaming it to an analytics platform. It monitors z/OS log data and SMF data (including z/OS Connect EE audit logs, SMF data record type 123) and forwards it to the configured destination. A web-based configuration tool is used to specify what data you want to collect from your z/OS system, where you want the data to be sent, and what form you want the data to arrive in at its destination. This configuration information is contained in a policy.

This chart shows a sample Splunk dashboard that provides a view of a z/OS Connect EE API workload.

The **API Requests** chart shows the number of requests per minute for the different APIs of the workload (catalog, customer and phonebook APIs). The search string for this chart is shown below:

```
* sourcetype="zos-smf_123*" |  timechart count(SM123SSI) by API_SERVICE_NAME
```

The sourcetype defines the type of data being searched, timechart defines the type of chart, SM123SSI represents the type of subsystem (z/OS Connect EE in this case) and records are counted by the API-SERVICE-NAME which represents the name of the API.

The **customer API by HTTP method** chart shows the different HTTP methods used to invoke the customer API.

The **catalog API by version** chart shows a breakdown of requests for the different versions of the catalog API. This chart allows us to monitor whether older versions of an API are still being used.

The **catalog API by channel** chart shows a breakdown of requests for the catalog API across different channels. This chart gives us a view of API requests across the Call Center, Mobile, Partner and Web channels. We use the request user ID to distinguish between the different channels.

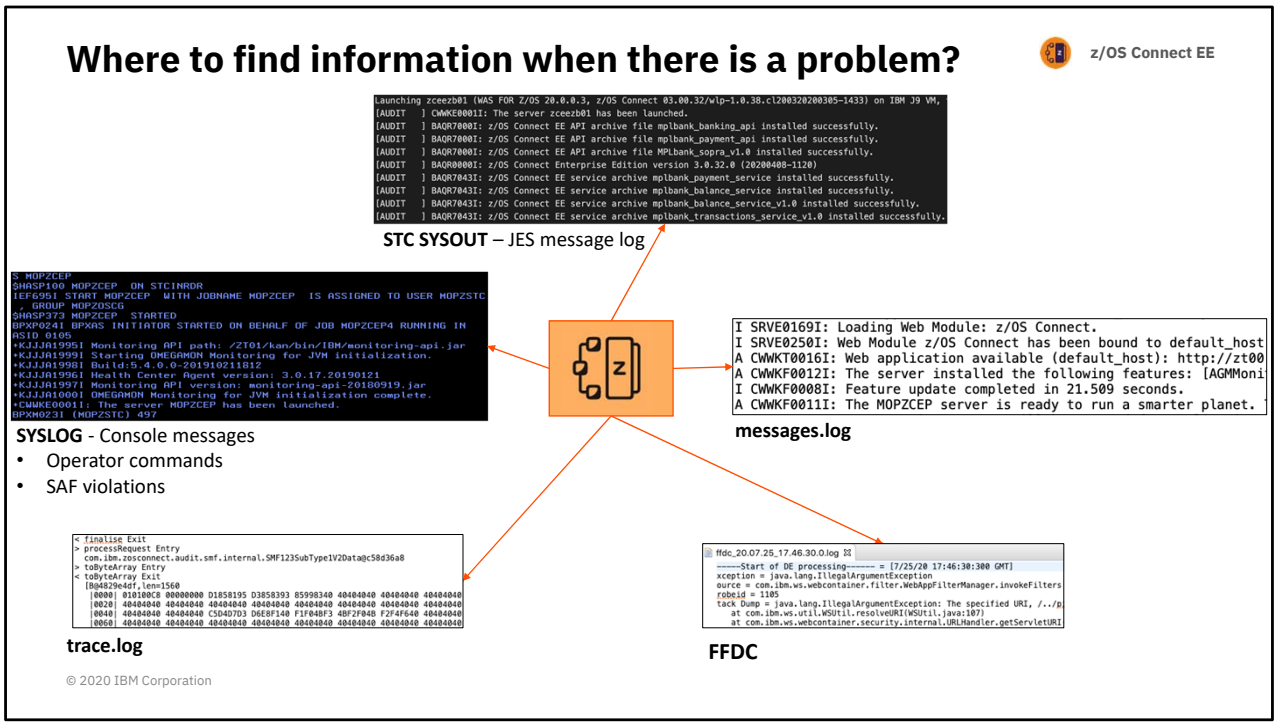
Top Tip!

The ability to identify performance issues quickly and to take corrective actions is a primary requirement for IT Operations

AVOID BLIND SPOTS!

/problem determination

Messages, logs and trace options



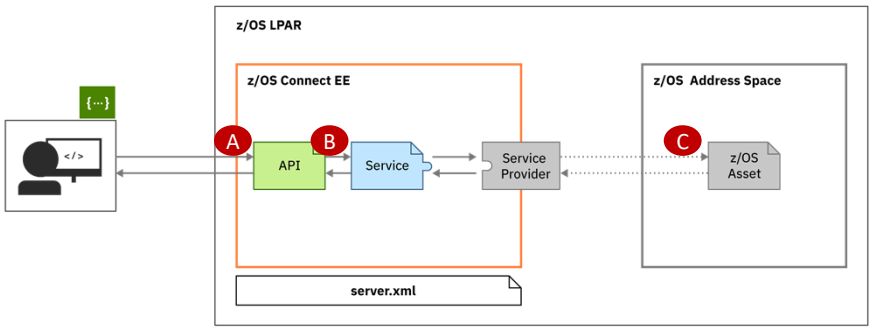
There are multiple places to look for information when a problem occurs:

1. **SYSLOG** – Console messages are written to SYSLOG and are intended for direct human consumption or automated operations. Note that you can use the Liberty zOSLogging element to redirect certain z/OS Connect EE messages to SYSLOG. Specify the wtoMessage attribute with a comma-separated list of message IDs to be written. For more information see **https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.liberty.autogen.zos.doc/ae/rwlp_config_zosLogging.html**
2. **SYSOUT** – Output of the STC. A subset of these messages are also written to SYSLOG. The granularity of messages is controlled by the consoleLogLevel attribute. The valid values are INFO, AUDIT (default), WARNING, ERROR, and OFF.
3. **messages.log** - This file contains all messages that are written or captured by the logging component: it contains detailed server log for API requests. All messages that are written to this file contain additional information such as the message time stamp and the ID of the thread that wrote the message. This file does not contain messages that are written directly by the JVM process. The location of messages.log is controlled by the logDirectory attribute (default location is WLP_OUTPUT_DIR/serverName/logs by default). You can use the DD statement **//MSGLOG DD SYSOUT=*** to redirect messages to the STC SYSOUT.

4. **FFDC** – First Failure Data Capture records include the exception stack and optional additional data that is recorded when an unexpected exception is caught by z/OS Connect EE
5. **trace.log** – Tracing is normally only enabled in development servers or at the request of IBM Service. You set logging options by adding a logging element to the server configuration file e.g <logging traceSpecification="SSL=all:SSLChannel=all:com.ibm.wsspi.webcontainer*=all"/>. For a list of z/OS Connect EE trace components see **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/troubleshooting/trace.html**
   The location of the trace file is controlled by the traceFileName attribute (default location is WLP_OUTPUT_DIR/serverName/logs)

Debugging common problems

z/OS Connect EE



A

- Connectivity
- TLS handshake
- HTTP 302 - use of HTTPS required
- HTTP 404 - incorrect path
- HTTP 401 - unauthenticated user
- HTTP 403 - unauthorized user

B

- HTTP 503 – service unavailable
- HTTP 400 - Missing mandatory parameter/ bad request
- Error in data mapping
- Subsystem reference not defined

C

- Connectivity
- TLS handshake
- Unauthenticated user
- Unauthorized user
- Malformed data

This illustrates common problems encountered with API provider. This is not an exhaustive list.

When trying to debug a problem with z/OS Connect EE, it is crucial to know where the error is coming from.
One would generally check the following elements:
1/ HTTP status code returned by the call & returned JSON response
2/ z/OS Connect EE messages.log
3/ z/OS Connect EE STDOUT
3/ z/OS syslog
4/ Subsystem log

The error can be of different nature: network, security, z/OS Connect EE application, subsystem application. For each category, there are certain tests that can be performed or messages to check.

Network: ping test, hostname resolution test, check TCP port binding, use network sniffing tool, verify TLS handshake (verify SSL configuration, trust store, key store, certificate expiration)

Security: check messages.log, check SAF messages in STDOUT or syslog, check subsystem log, verify provided credentials, verify access rights

z/OS Connect EE: check messages.log, enable specific traces to validate data, check service project and API project

Subsystem: check messages.log, check subsystem log, debug transaction with breakpoints

For more information on problem determination see
https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/troubleshooting/troubleshooting.html

Top Tip!

z/OS Connect EE

Look out for common problems

- Configuration errors?
  - Check configuration spelling
  - Check if XML syntax is correct
  - Check error/warning messages
  - Check default values
  - Check `updateTrigger` values i.e. how modifications are picked up
- Slow response times?
  - Are there any traces enabled ?
  - Is polling enabled ?
  - Are connections persisted ?
  - Consider using AsyncIO

This illustrates common problems related to configuration and slow response time.

Regarding ignored configuration :
1) When an element name, an attribute or a value is misspelled it is ignored.
2) When editing the server.xml it is recommended to use an XML editor like the z/OS Explorer as it would help to visualize the configuration with syntax coloring and highlight XML elements that are not properly opened/closed, missing double quotes for values, comments.
3) When attributes are not set, it doesn't mean that they are not used but rather that a default value has been used. These default values can be found on the z/OS Connect EE and Liberty Knowledge Center pages.

Regarding slow response time :
1) When Java or Liberty debugging are configured and print traces, it does impact the responsiveness of the z/OS Connect EE server
2) *Polled* is one of the values that the updateTrigger attribute can take, it implies that the server would frequently check if the monitored resources have been modified
3) By default, the z/OS Connect EE server does persist the connections with clients. The number of HTTP requests that can be used in a connection and the duration during which the connection can stay idle can both be configured. Make sure that the clients

are also configured to persist the connections

4) The Liberty server can be authorized to use z/OS native asynchronous TCP/IP sockets to improve performance and be more scalable. See **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/performance/performance_best_practices.html#performance_best_practices__asyncio**

For more information performance see **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/performance/performance_best_practices.html**

/performance

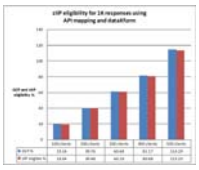
Measuring API performance

What are the main factors that impact performance?

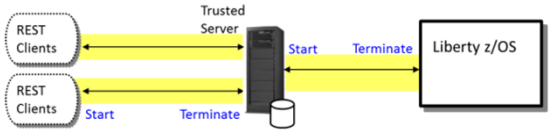
z/OS Connect EE

- Length and complexity of JSON messages

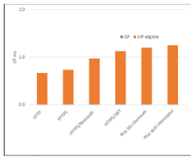
- zIIP availability



- Persistent connections



- Security
  - TLS
  - Client authentication
  - Authorization



This chart is a summary of some of the main performance considerations for z/OS Connect EE:

- As the length of message and number of array elements in a message increases, so will the response time the CPU processing time required to handle the request.
  Recommendation: Use the API toolkit to reduce the size of your JSON payload by excluding unused fields or renaming fields.
  Recommendation: If your payload contains arrays, optimize the JSON payload by using OCCURS DEPENDING ON (ODO) in COBOL, and REFER in PL/I, or use array counters, or both.
  For more information, see
  https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/performance/performance_payload_size.html

- z/OS Connect EE is a Java-based product and typically over 99% of MIPs are eligible for zIIP offload.
  Recommendation: Use performance data to monitor zIIP offload and to ensure sufficient zIIP capacity.

- When HTTPS is used for inbound requests to z/OS Connect EE, the use of persistent

connections outperform the use of non-persistent connections. When using persistent connections, the client can reuse the underlying socket connection and the TLS handshake is avoided.
Recommendation: Configure persistent sessions by setting the *keepAliveEnabled, maxKeepAliveRequests* and *persistTimeout* attributes on the *httpOptions* element in server.xml.

- Often security is at odds with performance, because the most secure techniques often involve the most processing overhead. The type of security model used can have a significant impact on CPU cost.
  Recommendation: By default, when https is used and client authentication is enabled, z/OS Connect EE attempts to map the provided client certificate to a user ID. This mapping can be expensive. Third-party authentication (for example with a JWT) can override the client certificate mapping which will provide better performance.

For more information on performance best practices, see **https://www.ibm.com/support/knowledgecenter/SS4SVW_3.0.0/performance/performance_overview.html**

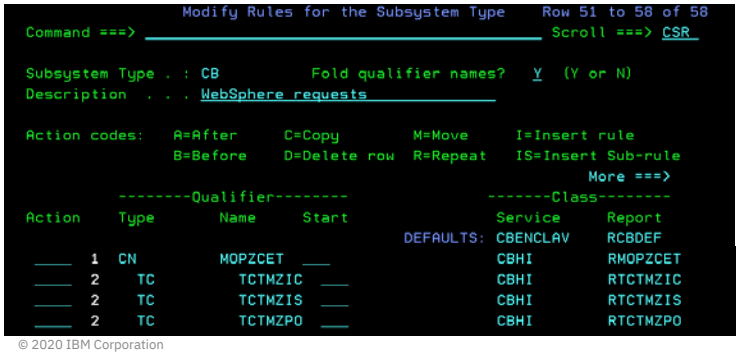
Download the z/OS Connect EE performance reports here **ibm.biz/zosconnect-performance-report**

Classifying API requests with WLM

z/OS Connect EE

```
<wlmClassification>
 <httpClassification transactionClass="TCTMZIC" method="GET"
              resource="/catalogManager/items?startItemID*" />
 <httpClassification transactionClass="TCTMZIS" method="GET"
              resource="/catalogManager/items/*" />
 <httpClassification transactionClass="TCTMZPO" method="POST"
              resource="/catalogManager/orders" />
</wlmClassification>
```

Assign API requests to transaction classes



Configure WLM service and report classes

61

*bin directory in service projects should not be stored in SCM.

A *transaction class* provides a granular approach for classifying API requests. For instance, you can classify requests for different API operations inside the same z/OS Connect EE server depending on the request URI.

A transaction class for z/OS Connect EE is enabled using the **zosWlm-1.0** feature. After adding this feature, validate that the z/OS Connect EE server is authorized to use the feature by checking for the message '*CWWKB0103I: Authorized service group ZOSWLM is available*' in the messages log.

Transaction classes are defined in the server.xml file. Example 1 shows an example classification of 3 API operations for the sample *catalog* API. we assign transaction class **TCIC** to the inquire catalog API operation which is invoked using an HTTP GET request with URI **/catalogManager/items**. We also assign transaction classes to the inquire single and place order API operations.

API requests received by a z/OS Connect EE server can then be classified using the **CB** classification type. The processor activity of z/OS Connect EE transactions is reported to WLM and RMF and so can be measured in RMF Workload Activity reports for service classes and report classes.

Recommendation: Define at least one transaction class for z/OS Connect EE API requests, so that you can differentiate between CPU consumption related to API activity (e.g data mapping) and CPU consumption related to server activity (e.g TLS and garbage collection).

See the following blog for more information on classifying API requests using WLM:
https://developer.ibm.com/mainframe/docs/managing-api-workloads/measuring-api-workloads/measuring-api-workloads-wlm/

Top Tip!

Classify API requests using WLM

This is INVALUABLE for performance analysis

Even if you use a single classification rule for all API requests

Classify API requests using WLM so that you have information on:

- Number of API requests
- Response times
- CPU consumption
- Whether CPU is being consumed on zIIPs or GPs
- Whether CPU is being consumed by API processing or server tasks (TLS, GC etc.)

Measuring API performance with RMF

z/OS Connect EE

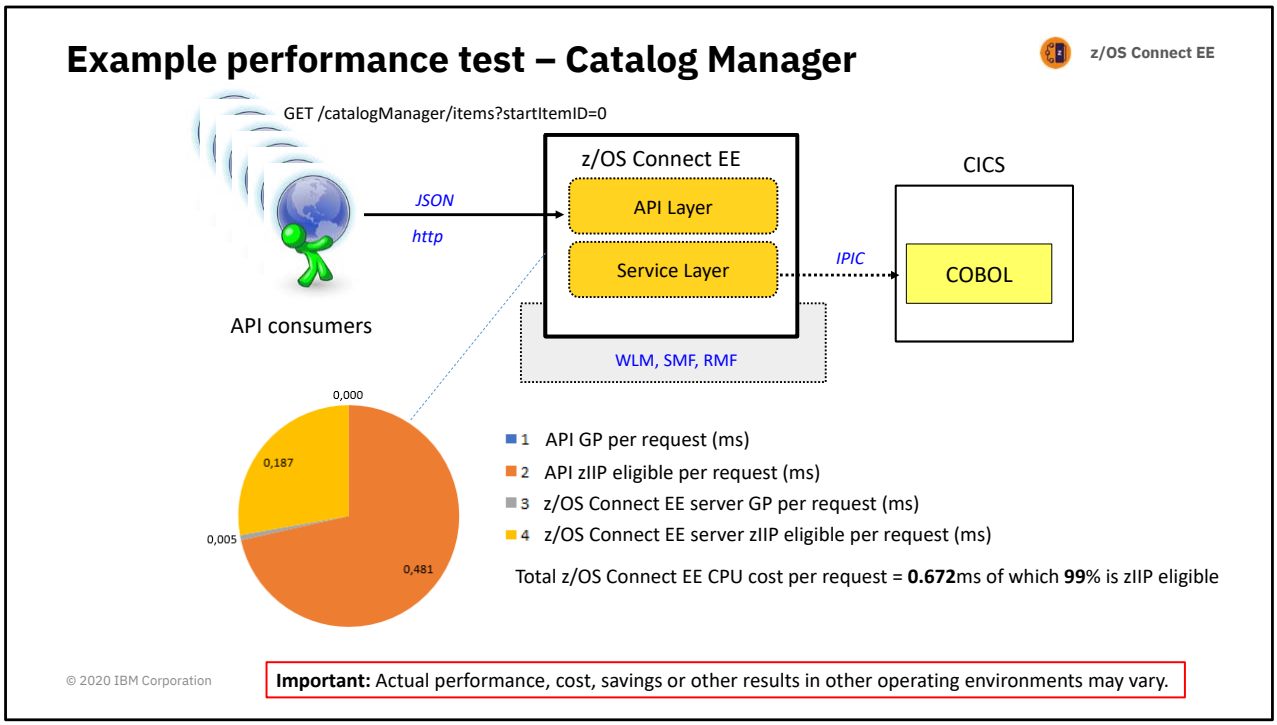
```
REPORT BY: POLICY=WLMPOL  REPORT CLASS=RTCTMZIC
                          DESCRIPTION =CB RC for inquire catalog API

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  SERVICE TIME  ---APPL %---
AVG       0.27  ACTUAL                 5  CPU    19.733 CP      3.29
MPL       0.27  EXECUTION              5  SRB     0.000  AAPCP   0.00
ENDED    30390  QUEUED                 0  RCT     0.000  IIPCP   3.17
END/S    50.65  R/S AFFIN              0  IIT     0.000
£SWAPS       0  INELIGIBLE             0  HST     0.000  AAP       N/A
EXCTD        0  CONVERSION             0  AAP       N/A  IIP       N/A
AVG ENC   0.27  STD DEV                0  IIP       N/A
```

- An average of **50.65** inquire catalog API requests were executed per second during the recording interval (10 minutes)
- The average response time for each request is **5**ms which includes the CICS transaction response time
- A total of **19.733** seconds of CPU time was consumed by the API
- The total CPU consumed is equivalent to **3.29**% of a CP of which **3.17**% is zIIP eligible

Note: The IIPCP value represents the percentage of a CP that could have run on a zIIP if a zIIP had been available on the system.

This chart shows an extract of an example RMF Workload Activity report for the report class **RTCTMZIC**. It shows:

- An average of **50.65** inquire catalog API requests were executed per second during the recording interval (10 minutes)
- The average response time for each request is **5**ms which includes the CICS transaction response time
- A total of 19.733seconds of CPU time was consumed by the API during the reporting interval (10 minutes).
- The total CPU consumed by the API is equivalent to 3.29% of a CP, out of which 3.17% of the CP is zIIP eligible.



This chart shows some example performance measurements.

Given that the z/OS Connect EE server was only running the catalog API at the time of the measurement, we can measure the CPU cost of the API by adding the CPU for the API report class **RCBTCIC** to the CPU for the z/OS Connect EE server report class **RSTCZCEL** .

The pie chart shows the breakdown of CPU usage across these two report classes and separated into GP ms and zIIP eligible ms.

Important: the test machine used for these tests did not have zIIPs configured.

Recommendations: If you are experiencing performance problems, consider the following potential optimizations:

- Disable polling. If you need to update your server.xml file, use the Modify refresh command
- Configure persistent connections using the keepAliveEnabled, maxKeepAliveRequests and persistTimeout attributes on the httpOptions element in server.xml

- Configure hardware cryptography if a large number of TLS handshakes are occuring
- Minimise JSON message lengths and complexity

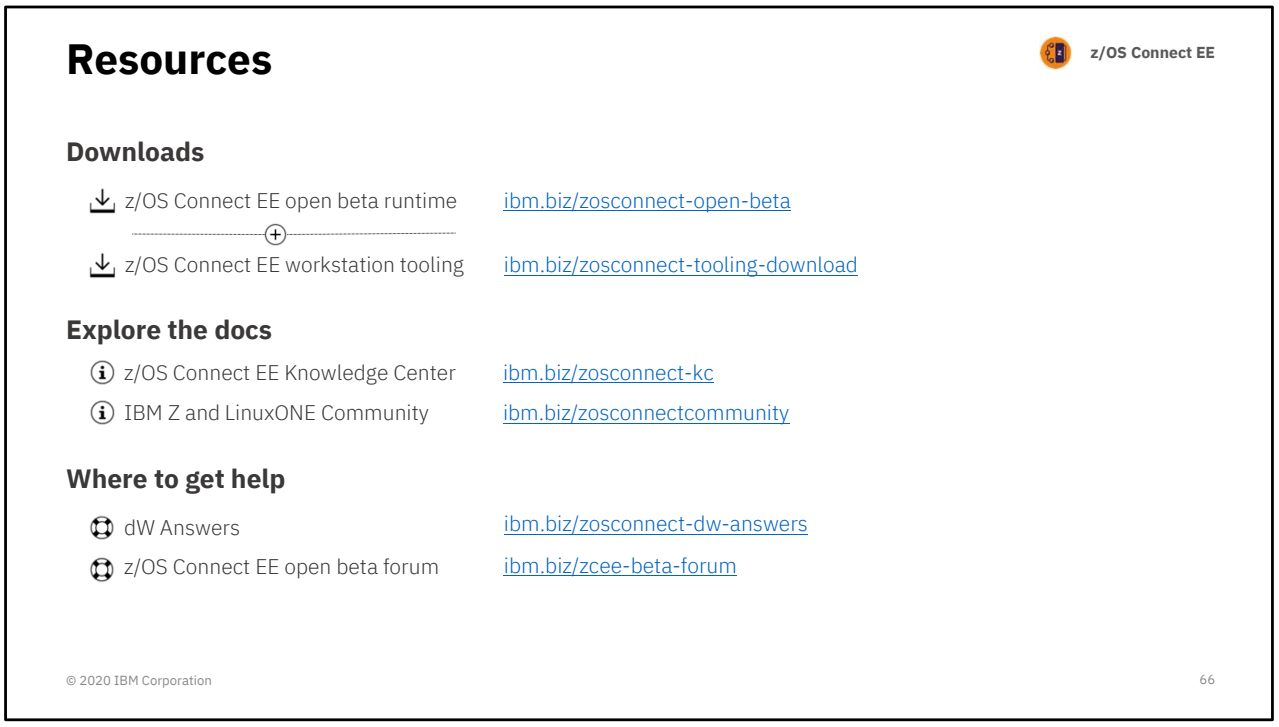
/questions?

Please contact the authors of this presentation if you have questions:

nigel_williams@uk.ibm.com
aymeric.affouard@fr.ibm.com
eric.phan@fr.ibm.com

Thanks to the following people for their contributions to this presentation:

- Sue Bayliss
- Demelza Farrer
- Alan Hollingshead
- Mitch Johnson
- Edward McCarthy
- Anthony Papageorgiou
- Kate Robinson
- Kenishia A Sapp



Here is a list of useful resources for z/OS Connect EE.

(Don’t forget to download the runtime *and* the workstation tooling.)

dW Answers and the open beta forums are regularly monitored by the development team.