

Course Guide

Create, Secure, and Publish APIs with IBM API Connect v2018

Course code WD514 / ZD514 ERC 6.0



Contents

Trademarks	xii
Course description	xiii
Agenda	xv
Unit 1. IBM API Connect V2018 overview.....	1-1
Unit objectives	1-2
1.1. API Connect overview	1-3
API Connect overview	1-4
Topics	1-5
API terminology	1-6
What is API Connect?	1-7
API Connect key capabilities	1-8
Create and run APIs	1-9
Secure and control APIs	1-10
Manage APIs	1-11
Socialize APIs	1-12
Analyze APIs	1-13
Manage APIs through the API lifecycle	1-14
What is the API Connect Cloud?	1-15
Cloud topology	1-16
Defining the cloud topology: Cloud Manager	1-18
Cloud Manager tasks	1-19
API Connect: Other user interfaces and user roles	1-21
1.2. Gateway options.....	1-23
Gateway options	1-24
Topics	1-25
Security and control with industry-leading DataPower API gateway	1-26
DataPower in the cloud	1-27
Downloading DataPower Gateway for Developers on Docker	1-28
API gateway types	1-30
1.3. Software and hardware prerequisites.....	1-32
Software and hardware prerequisites	1-33
Topics	1-34
API Connect: Requirements	1-35
Unit summary	1-36
Review questions	1-37
Review answers	1-38
Exercise: Review the API Connect development and runtime environment	1-39
Exercise objectives	1-40
Unit 2. API Connect development platform	2-1
Unit objectives	2-2
What is the IBM API Connect developer toolkit?	2-3
API Connect toolkit options	2-4
Toolkit prerequisites	2-5
Review the supported level for Node.js	2-6
Verify the toolkit installation	2-7
API Connect Toolkit: The apic lb command	2-8

API Designer	2-9
How to create APIs in API Connect	2-10
Unit summary	2-11
Review questions	2-12
Review answers	2-13
Unit 3. Creating an API definition	3-1
Unit objectives	3-2
What is an API definition?	3-3
Open API: REST API interface standard	3-4
OpenAPI definition structure	3-5
Sample OpenAPI definition	3-7
How do you create an API definition? (1 of 2)	3-8
How do you create an API definition? (2 of 2)	3-9
OpenAPI definition: Name, version, and licensing	3-10
Parts of an REST API definition	3-11
Parts of an REST API operation	3-13
Review: HTTP methods in REST architecture	3-14
Example: Savings plan estimate	3-15
Example: GET /estimate	3-16
Step 1: Define a path, and operation	3-17
Step 2: Define the result data type	3-18
Step 3: Define the input parameters and response message	3-19
Example: POST /estimate	3-20
Step 1: Create a POST operation in the existing path	3-21
Step 2: Define the plan schema data type	3-22
Step 3: Define the input parameter in the message body	3-23
API definition: Source view	3-24
Example: savings.yaml	3-25
IBM extensions to the OpenAPI definition format	3-26
Extensions: Lifecycle settings	3-27
Extensions: Message processing policy assembly	3-28
Example: The invoke policy	3-29
Unit summary	3-30
Review questions	3-31
Review answers	3-32
Exercise: Create an API definition from an existing API	3-33
Exercise objectives	3-34
Unit 4. Defining APIs that call REST and SOAP services.....	4-1
Unit objectives	4-2
API runtime message flow	4-3
Scenario 1: Proxy existing APIs	4-4
Scenario 2: Implement APIs	4-5
What types of APIs can you define?	4-6
Scenarios	4-7
4.1. Proxy an existing REST API.....	4-8
Proxy an existing REST API	4-9
Topics	4-10
Scenario one: Define a REST API to proxy to an existing REST service	4-11
Scenario one: Gateway forwards REST API requests	4-12
Step 1: Create a REST API definition in API Manager	4-13
Step 2: Define the base path and target endpoint	4-14
Step 3: Edit the generated OpenAPI definition	4-15
Step 4: Select the gateway type	4-16
Step 5: Add definitions for response data type	4-17

Step 6: Add a path and operations	4-18
Step 7: Configure the gateway to call the REST application	4-19
Step 8: Test the API definition	4-20
4.2. Proxy an existing SOAP API.....	4-21
Proxy an existing SOAP API	4-22
Topics	4-23
Scenario two: Define a SOAP API	4-24
Scenario two: Gateway forwards SOAP API requests	4-25
Step 1: Review the existing SOAP service	4-26
Step 2: Create a SOAP API definition in API Manager	4-27
Step 3: Review the API operations	4-28
Step 4: Review the request and response type definitions	4-29
Step 5: Review the invoke policy in the assembly flow	4-30
Step 6: Test the SOAP API in the API Manager	4-31
Step 7: Response message from the SOAP API call	4-32
Unit summary	4-33
Review questions	4-34
Review answers	4-35
Exercise: Define an API that calls an existing SOAP service	4-36
Exercise objectives	4-37
Unit 5. Implementing APIs with the LoopBack framework.....	5-1
Unit objectives	5-2
5.1. Introduction to the LoopBack framework	5-3
Introduction to the LoopBack framework	5-4
Topics	5-5
Example: Product inventory and price calculator	5-6
Message flows in IBM API Connect	5-7
What is LoopBack?	5-8
LoopBack data source connectors	5-9
LoopBack database connectors	5-10
LoopBack connectors to back-end systems	5-11
Implement an API with a NodeJS LoopBack application	5-12
5.2. Implement an API with a LoopBack application	5-13
Implement an API with a LoopBack application	5-14
Topics	5-15
Create a LoopBack application: example	5-16
Step 1: Generate a LoopBack application	5-17
Step 2: Define a data source	5-18
Step 3: Configure the data source	5-19
Step 4: Create models, properties, and relationships	5-21
Step 5: Implement API logic with a remote method	5-22
Step 6: Start the LoopBack application and Explorer	5-24
Step 7: Review the application in the Loopback Explorer	5-25
Step 8: Test the operation in the Loopback Explorer	5-26
Step 9: Review the results	5-27
Relationship between API gateway and LoopBack	5-28
Unit summary	5-29
Review questions	5-30
Review answers	5-31
Exercise: Create a LoopBack application	5-32
Exercise objectives	5-33
Unit 6. LoopBack models, properties, and relationships.....	6-1
Unit objectives	6-2
6.1. LoopBack framework in API Connect.....	6-3

LoopBack framework in API Connect	6-4
Topics	6-5
LoopBack: An open source Node.js API framework	6-6
LoopBack framework: Models, properties, relationships	6-7
LoopBack framework: API mapped to models	6-8
LoopBack framework: Data persistence with connectors	6-9
LoopBack framework: Non-database connectors	6-10
LoopBack framework: Remote methods and hooks	6-11
LoopBack framework: Packaging	6-12
The role of LoopBack in API Connect	6-13
LoopBack applications in an API Connect solution	6-14
Example: Note sample application	6-15
Example: Note model	6-16
How to implement an API with the LoopBack framework	6-18
Next steps after you implement the API	6-19
6.2. LoopBack models, properties, and relationships	6-20
LoopBack models, properties, and relationships	6-21
Topics	6-22
Steps: LoopBack application, models, properties, relations	6-23
Generate a LoopBack application with apic loopback	6-24
Example: Create a LoopBack application	6-25
LoopBack application: Directory structure	6-26
Example: Create a model and properties	6-27
Where does LoopBack store model information?	6-28
LoopBack model inheritance	6-29
LoopBack properties: Data types	6-30
Example: Model definition file	6-31
Example: Model script file	6-32
Example: Bind the model to a data source	6-33
How do you define a relationship between models?	6-34
Example: Create model relations	6-35
Example: Relations in the model definition file	6-36
Model relation types	6-37
Unit summary	6-38
Review questions	6-39
Review answers	6-40
Unit 7. Defining data sources with connectors	7-1
Unit objectives	7-2
7.1. LoopBack data sources	7-3
LoopBack data sources	7-4
Topics	7-5
Steps: LoopBack data sources	7-6
What is a LoopBack connector?	7-7
How do you persist model data with connectors?	7-8
Database connectors	7-9
Memory connector	7-10
Example: Create a memory data source	7-11
Example: Data sources configuration file	7-12
Example: Model configuration file	7-13
Non-database connectors	7-14
7.2. Generating a LoopBack application that uses an in-memory connector	7-15
Generating a LoopBack application that uses an in-memory connector	7-16
Topics	7-17
Create a LoopBack local file application: example	7-18
Step 1: Generate a LoopBack application	7-19

Step 2: Define a data source	7-20
Step 3: Configure the data source persistence file	7-21
Step 4: Create the models and properties	7-22
Step 5: Change the default id injection	7-23
Step 6: Start the LoopBack application and Explorer	7-24
Step 7: Review the application in the Loopback Explorer	7-25
Step 8: Test the operation in the Loopback Explorer	7-26
Step 9: Review the results	7-27
7.3. Generating LoopBack models from discovery and introspection	7-28
Generating LoopBack models from discovery and introspection	7-29
Topics	7-30
Model discovery and introspection	7-31
Generate models from relational databases	7-32
Create model instances from unstructured data	7-34
Example: Create a model from introspection (1 of 2)	7-35
Example: Create a model from introspection (2 of 2)	7-36
Unit summary	7-37
Review questions	7-38
Review answers	7-39
Exercise: Define LoopBack data sources	7-40
Exercise objectives	7-41
Unit 8. Implementing remote methods and event hooks	8-1
Unit objectives	8-2
LoopBack framework: Remote methods and hooks	8-3
Steps: LoopBack remote methods and hooks	8-4
Adding logic to models	8-5
What is a remote method?	8-6
Example: Remote method	8-7
What is a remote hook?	8-8
Example: Remote hook	8-9
What is an operation hook?	8-10
Types of operation hooks	8-11
Which operations trigger operation hooks?	8-12
Example: Access operation hook	8-13
Example: After save operation hook	8-14
When do you implement remote methods?	8-15
When do you implement remote hooks?	8-16
When do you implement operation hooks?	8-17
Unit summary	8-18
Review questions	8-19
Review answers	8-20
Exercise: Implement event-driven functions with remote and operation hooks	8-21
Exercise objectives	8-22
Unit 9. Assembling message processing policies	9-1
Unit objectives	9-2
What is a message processing policy?	9-3
Message processing policies at run time	9-4
Assembly editor: Creating policy assemblies	9-5
Assembly editor: Palette and canvas	9-6
Assembly editor: Filter, search, and gateway type	9-7
Assembly editor: Magnify and zoom	9-8
Assembly editor: Properties editor	9-9
API policies and logic constructs	9-10
Example scenarios for policy assemblies	9-12

Example one: Forward an API call with the invoke policy	9-13
Example two: Switch case by API operation	9-14
Example three: Map multiple API calls into a response	9-15
Example: Define input parameters in a map policy	9-16
Example: Map API results into a single response message	9-17
Example four: Transform REST to SOAP	9-18
Example: Define input parameters in a map policy	9-19
Example: Map REST parameters to a SOAP request	9-20
What is the message payload?	9-21
Example five: Validate properties in an HTTP message	9-22
Example six: Store message payload in API analytics	9-23
Unit summary	9-25
Review questions	9-26
Review answers	9-27
Exercise: Assemble message processing policies	9-28
Exercise objectives	9-29
Unit 10. Declaring client authorization requirements	10-1
Unit objectives	10-2
10.1. API security concepts	10-3
API security concepts	10-4
Topics	10-5
Authentication and authorization: API security definitions	10-6
How do you secure your APIs in API Connect?	10-7
What types of security definitions can you define?	10-8
10.2. Identify client applications with API key	10-9
Identify client applications with API key	10-10
Topics	10-11
API key: A unique client application identifier	10-12
Example: Secure with Client ID (1 of 2)	10-13
Example: Secure with Client ID (2 of 2)	10-14
Example: Add client secret security definition	10-15
Applying security definitions (1 of 2)	10-16
Applying security definitions (2 of 2)	10-17
Rules for defining client ID and client secret	10-18
Example: Client ID and client secret in the message header	10-19
10.3. Authenticate clients with HTTP basic authentication	10-20
Authenticate clients with HTTP basic authentication	10-21
Topics	10-22
Verifying identity with HTTP basic authentication	10-23
Example: Storing credentials in HTTP request header	10-24
Setting up a user registry (1 of 3)	10-25
Setting up a user registry (2 of 3)	10-26
Setting up a user registry (3 of 3)	10-27
Example: Basic authentication security definition	10-28
Example: Apply basic authentication security to the API	10-29
10.4. Introduction to OAuth 2.0	10-30
Introduction to OAuth 2.0	10-31
Topics	10-32
What is OAuth?	10-33
Example: Allow third-party access to shared resources	10-35
Example: Third-party access to inventory API resources	10-36
OAuth Step 1: Resource owner requests access	10-37
OAuth Step 2: OAuth client redirection to owner	10-38
OAuth Step 3: Authenticate owner with authorization service	10-39
OAuth Step 4: Ask resource owner to grant access to resources	10-40

OAuth Step 5: Resource owner grants client access to resources	10-41
OAuth Step 6: Authorization service sends authorization grant code to client	10-42
OAuth Step 7: Client requests access token from authorization service	10-43
OAuth Step 8: Authorization server sends authorization token to client	10-44
OAuth Step 9: OAuth client sends access token to resource service	10-45
OAuth Step 10: Resource server grants access to OAuth client	10-46
Unit summary	10-47
Review questions	10-48
Review answers	10-49
Unit 11. Creating an OAuth 2.0 provider	11-1
Unit objectives	11-2
Role of IBM API Connect in the OAuth flow	11-3
What are the steps to secure an API with OAuth 2.0?	11-4
What is an OAuth Provider?	11-5
11.1. Create an OAuth Provider	11-6
Create an OAuth Provider	11-7
Topics	11-8
OAuth Provider types	11-9
Create an authentication registry (1 of 3)	11-10
Create an authentication registry (2 of 3)	11-11
Create an authentication registry (3 of 3)	11-12
Create a Native OAuth Provider (1 of 5)	11-13
Create a Native OAuth Provider (2 of 5)	11-14
Create a Native OAuth Provider (3 of 5)	11-15
Create a Native OAuth Provider (4 of 5)	11-16
Create a Native OAuth Provider (5 of 5)	11-17
Native OAuth Provider in Cloud Manager resources	11-18
OAuth Provider: OAuth flow and grant types	11-19
OAuth Provider: Client types	11-20
Configure the catalog to use the resources (1 of 2)	11-21
Configure the catalog to use the resources (2 of 2)	11-22
11.2. Secure an API with OAuth 2.0 authorization	11-23
Secure an API with OAuth 2.0 authorization	11-24
Topics	11-25
What is an OAuth 2.0 security definition?	11-26
Configure OAuth security settings for the API (1 of 3)	11-27
Configure OAuth security settings for the API (2 of 3)	11-28
Configure OAuth security settings for the API (3 of 3)	11-29
Enable OAuth security for the API	11-30
Unit summary	11-31
Review questions	11-32
Review answers	11-33
Exercise: Declare an OAuth 2.0 Provider and security requirement	11-34
Exercise objectives	11-35
Unit 12. Deploying an API to a Docker container	12-1
Unit objectives	12-2
What is Docker?	12-3
Cargo transport pre-1960	12-4
Solution: Cargo container	12-5
Why Docker containers	12-6
Why use Docker containers?	12-7
Virtual machines versus containers	12-8
Definition: Hypervisor	12-9
Containers on virtual machines	12-10

Docker terminology	12-11
Docker Engine	12-12
Docker architecture	12-13
Docker version	12-14
Docker CLI	12-15
Docker registry	12-16
Docker Hub	12-17
Dockerfile	12-18
Building images from a Dockerfile	12-19
Dockerfile example: Inventory application	12-20
Listing images	12-22
Run the application on the Docker container	12-23
Listing containers	12-24
Useful Docker commands	12-25
API runtime message flow: exercise case study	12-26
Unit summary	12-27
Review questions	12-28
Review answers	12-29
Exercise: Deploy an API implementation to a container runtime	12-30
Exercise objectives	12-31
Unit 13. Publishing and managing products and APIs	13-1
Unit objectives	13-2
Product, plan, API hierarchy	13-3
Define product and plan	13-4
API product definition file	13-5
Add a product (1 of 3)	13-6
Add a product (2 of 3)	13-7
Add a product (3 of 3)	13-8
Catalogs	13-9
Stage a product and API	13-10
Publish a product and API (1 of 2)	13-11
Publish a product and API (2 of 2)	13-12
Review the published products (1 of 2)	13-13
Review the published products (2 of 2)	13-14
Review from the terminal: Toolkit interface (1 of 2)	13-15
Review from the terminal: Toolkit interface (2 of 2)	13-16
Lifecycle of products and API resources	13-17
Stage and publish a previously published product (1 of 2)	13-18
Stage and publish a previously published product (2 of 2)	13-19
Manage published products in API Manager	13-20
Remove a product from the catalog (1 of 2)	13-21
Remove a product from the catalog (2 of 2)	13-22
Permissions for managing products	13-23
Unit summary	13-24
Review questions	13-25
Review answers	13-26
Exercise: Define and publish an API product	13-27
Exercise objectives	13-28
Unit 14. Subscribing and testing APIs	14-1
Unit objectives	14-2
Role of application developers	14-3
Self-registration on the Developer Portal (1 of 5)	14-4
Self-registration on the Developer Portal (2 of 5)	14-5
Self-registration on the Developer Portal (3 of 5)	14-6

Self-registration on the Developer Portal (4 of 5)	14-7
Self-registration on the Developer Portal (5 of 5)	14-8
Create an application (1 of 3)	14-9
Create an application (2 of 3)	14-10
Create an application (3 of 3)	14-11
Subscribe an application to a Product plan (1 of 6)	14-12
Subscribe an application to a Product plan (2 of 6)	14-13
Subscribe an application to a Product plan (3 of 6)	14-14
Subscribe an application to a Product plan (4 of 6)	14-15
Subscribe an application to a Product plan (5 of 6)	14-16
Subscribe an application to a Product plan (6 of 6)	14-17
Approval requests (1 of 3)	14-18
Approval requests (2 of 3)	14-19
Approval requests (3 of 3)	14-20
Testing an API in the Developer Portal	14-21
Testing an API in the Developer Portal (1 of 6)	14-22
Testing an API in the Developer Portal (2 of 6)	14-23
Testing an API in the Developer Portal (3 of 6)	14-24
Testing an API in the Developer Portal (4 of 6)	14-25
Testing an API in the Developer Portal (5 of 6)	14-26
Testing an API in the Developer Portal (6 of 6)	14-27
Status of applications in API Manager	14-28
Unit summary	14-29
Review questions	14-30
Review answers	14-31
Exercise: Subscribe and test APIs	14-32
Exercise objectives	14-33
Unit 15. Course summary	15-1
Unit objectives	15-2
Course objectives	15-3
To learn more on the subject	15-4
Enhance your learning with IBM resources	15-5
Earn an IBM Badge	15-6
Unit summary	15-7
Appendix A. List of abbreviations	A-1

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

Bluemix®

DB™

IBM Bluemix™

Cloudant®

Express®

IMS™

DataPower®

IBM API Connect™

Notes®

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

LoopBack® and StrongLoop® are trademarks or registered trademarks of StrongLoop, Inc., an IBM Company.

Social® is a trademark or registered trademark of TWC Product and Technology, LLC, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Course description

Create, Secure, and Publish APIs with IBM API Connect v2018

Duration: 4 days

Purpose

This course teaches you how to create, publish, and secure APIs with IBM API Connect V2018. You build Node.js API applications with the LoopBack framework. You define API interfaces according to the OpenAPI specification. You assemble message processing policies and define client authorization schemes, such as OAuth 2.0, in the API definition. You publish APIs and make them available on a secure gateway and on the Developer Portal.

Audience

This course is designed for API developers: software developers who define and implement API operations.

Prerequisites

Before taking this course, you should successfully complete *Developing REST APIs with Node.js for IBM Bluemix* (VY102G).

Objectives

- Create APIs in API Connect
- Implement APIs with the LoopBack Node.js framework
- Create message processing policies that transform API requests and responses
- Authorize client API requests with security definitions
- Enforce an OAuth flow with an OAuth 2.0 API security
- Publish, and test APIs on the API Connect cloud

Contents

- IBM API Connect V2018 overview
- Exercise: Review the API Connect development and runtime environment
- API Connect development platform
- Creating an API definition
- Exercise: Create an API definition from an existing API
- Defining APIs that call REST and SOAP services

- Exercise: Define an API that calls an existing SOAP service
- Implementing APIs with the LoopBack framework
- Exercise: Create a LoopBack application
- LoopBack models, properties, and relationships
- Defining data sources with connectors
- Exercise: Define LoopBack data sources
- Implementing remote methods and event hooks
- Exercise: Implement event-driven functions with remote and operation hooks
- Assembling message processing policies
- Exercise: Assemble message processing policies
- Declaring client authorization requirements
- Creating a Native OAuth Provider
- Exercise: Declare an OAuth 2.0 provider and security requirement
- Deploying an API to a Docker container
- Exercise: Deploy an API implementation to a container runtime environment
- Staging, publishing, and deploying an API product
- Exercise: Define and publish an API product
- Subscribing and testing APIs
- Exercise: Subscribe and test APIs
- Troubleshooting
- Course summary

Agenda



Note

The following unit and exercise durations are estimates, and might not reflect every class experience.

Day 1

- (00:15) Course introduction
- (01:00) Unit 1. IBM API Connect V2018 overview
- (00:45) Exercise 1. Review the API Connect development and runtime environment
- (00:30) Unit 2. API Connect development platform
- (01:00) Unit 3. Creating an API definition
- (01:00) Exercise 2. Create an API definition from an existing API
- (01:00) Unit 4. Defining APIs that call REST and SOAP services
- (00:30) Exercise 3. Define an API that calls an existing SOAP service

Day 2

- (01:00) Unit 5. Implementing APIs with the LoopBack framework
- (00:45) Exercise 4. Create a LoopBack application
- (01:15) Unit 6. LoopBack models, properties, and relationships
- (01:00) Unit 7. Defining data sources with connectors
- (01:00) Exercise 5. Define LoopBack data sources
- (00:45) Unit 8. Implementing remote methods and event hooks
- (00:45) Exercise 6. Implement event-driven functions with remote and operation hooks

Day 3

- (01:00) Unit 9. Assembling message processing policies
- (01:30) Exercise 7. Assemble message processing policies
- (01:00) Unit 10. Declaring client authorization requirements
- (00:45) Unit 11. Creating an OAuth 2.0 provider
- (01:00) Exercise 8. Declare an OAuth 2.0 provider and security requirement

Day 4

- (00:45) Unit 12. Deploying an API to a Docker container
- (00:30) Exercise 9. Deploy an API implementation to a container runtime environment
- (01:00) Unit 13. Publishing and managing products and APIs
- (00:30) Exercise 10. Define and publish an API product
- (01:00) Unit 14. Subscribing and testing APIs
- (01:00) Exercise 11. Subscribe and test APIs
- (00:05) Unit 15. Course summary

Unit 1. IBM API Connect V2018 overview

Estimated time

01:00

Overview

This unit explains the scope and purpose of IBM API Connect V2018 from the perspective of an API developer and cloud administrator. You review the key capabilities of API Connect. You examine the nature of an on-premises cloud and how the cloud is configured in the API Cloud Manager user interface. You review the different gateway types for securing and managing APIs. Identify the requirements for installing an on-premises API Connect cloud.

How you will check your progress

- Review questions
- Lab exercise

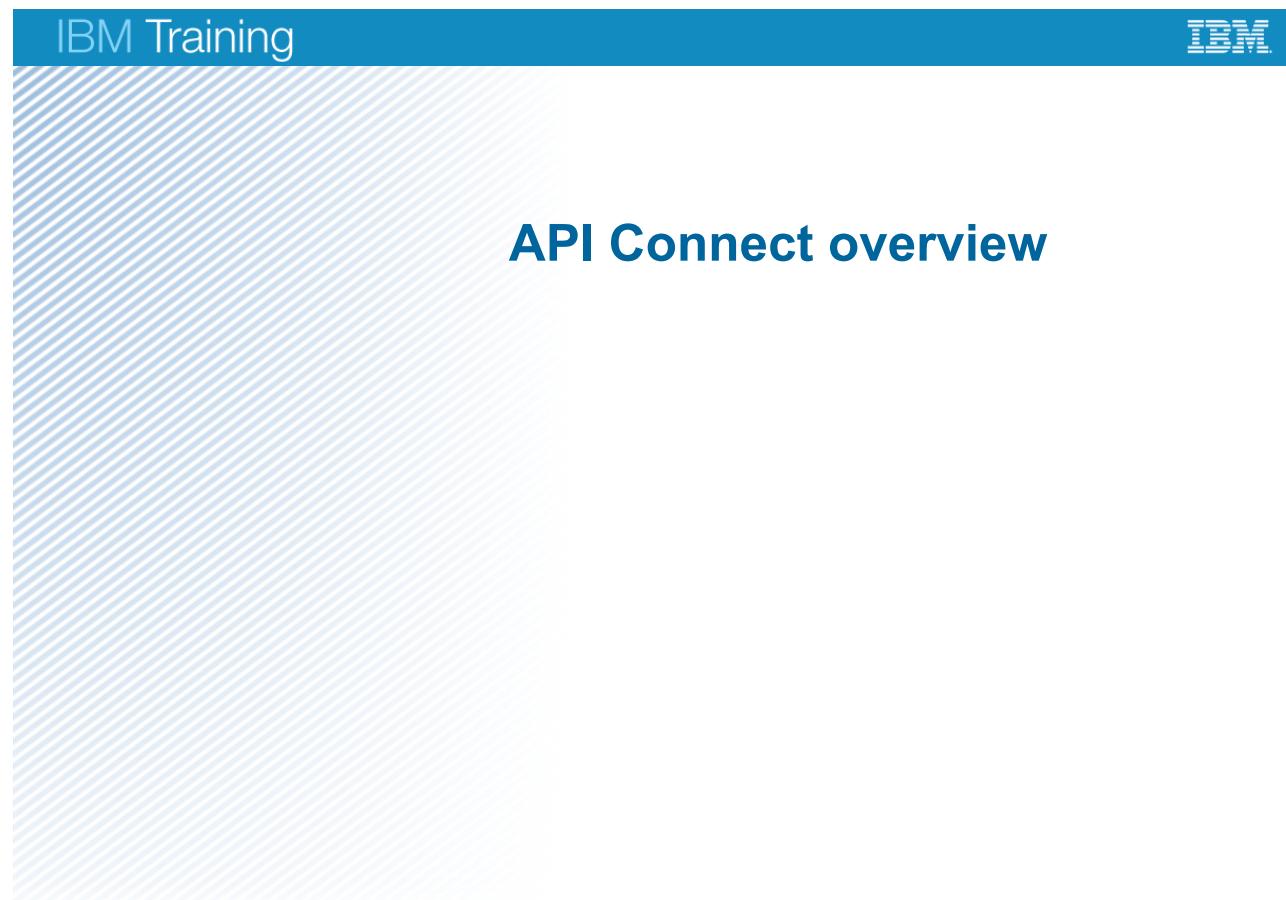
References

https://www.ibm.com/support/knowledgecenter/SSMNED_2018/mapfiles/getting_started.html

Unit objectives

- Explain what is IBM API Connect
- Describe the key capabilities of API Connect
- Describe how API Connect manages API through the entire API lifecycle
- Identify the components of an API Connect on-premises cloud
- Describe the use of the Cloud Manager user interface to administer the cloud topology and resources
- Describe the different gateway types for securing and managing APIs
- Identify the requirements for installing an API Connect on-premises cloud

1.1. API Connect overview



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-2. API Connect overview

Topics

API Connect overview

- Gateway options
- Software and hardware prerequisites

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-3. Topics

API terminology

- What is an **API**?
 - An application programming interface is a collection of remote service operations that you make available to API consumers
 - In IBM API Connect, the API is a collection of REST service operations
 - Representation State Transfer (REST) is an architectural style that uses a simple set of HTTP-based transactions, such as GET, POST, PUT, and DELETE
- What is an **API consumer**?
 - An API consumer is an application that **calls** remote operations in an API
- What is an **API provider**?
 - An API provider is an application or a system that **implements** the REST service operation in an API
- What is an **API gateway**?
 - An API gateway manages access to a set of API operations
 - The gateway enforces service policies to restrict consumer access to APIs

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-4. API terminology

Before you learn about the solution architecture for IBM API Connect, it is important to define the roles in an organization that publishes a set of APIs for its clients. The term “application programming interface (API)” is used in many areas of software development. In the context of IBM API Connect, an **API** is a collection of service operations that you make available on a network. The clients that call these API operations are known as **API consumers**. The organization or company that makes a set of services available is the **API provider**.

The API gateway is between the API consumer and the API provider. This gateway server or network appliance mediates and regulates requests to the posted API services. It enforces a set of rules, or services policies, that define how API consumers can access the API.

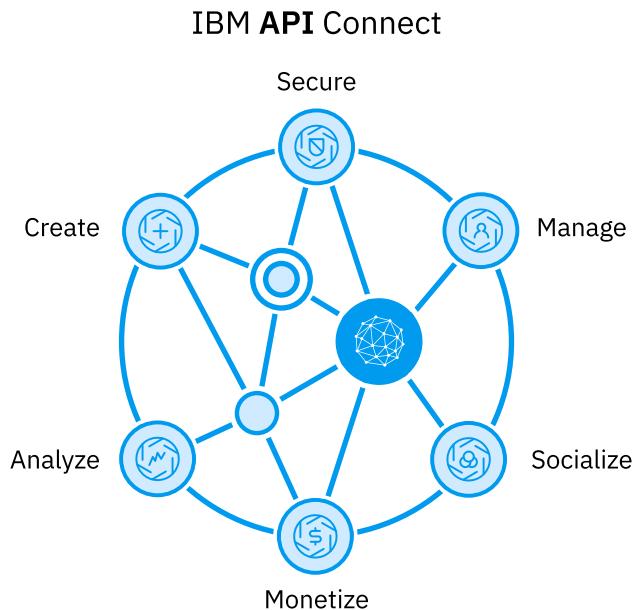
What is API Connect?

- An integrated solution that includes creating, running, managing, and securing APIs for a range of applications in a digital environment
- What does API Connect provide?
 - Automated, visual, and coding options for creating APIs
 - Node.js Loopback framework support for generating API implementations
 - Creation of models and APIs from diverse data sources
 - Lifecycle and governance for APIs, products, and plans
 - Advanced API usage analytics
 - Customizable, self-service Developer Portal for publishing APIs
 - Policy enforcement, security, and control provided by the IBM DataPower Gateway

Figure 1-5. What is API Connect?

IBM API Connect is an integration solution for enterprise grade APIs. The solution consists of a set of components for API creation, API runtime management, API security, and control. The focus of IBM API Connect is to build a modern system where APIs are integration point for digital applications: mobile, web, Business Partner applications, and internal applications.

API Connect key capabilities



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

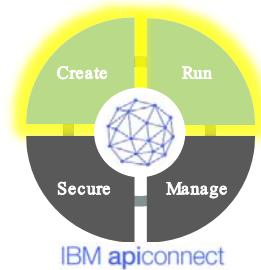
Figure 1-6. API Connect key capabilities

IBM API Connect is a comprehensive API solution that focuses on all stages of an API lifecycle: API creation, API security, API publishing, API metrics, and lifecycle management.



Create and run APIs

- Write APIs in Node.js
- Loopback open source framework
 - Develop and test by using a rich UI or CLI
 - Rapidly expose a wide variety of data sources as REST APIs
 - Build APIs code-first or by using model-driven development
 - Automatic generation of API models from a range of data sources
- Local API development and testing
- Visual API composition with the API Manager or API Designer
- Runtime management that uses a choice of container orchestration software
 - Docker, Kubernetes, IBM Cloud Private



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-7. Create and run APIs

IBM API Connect includes a set of rapid model-driven API creation tools. The LoopBack framework is an open source Node.js project that maps business models to a set of data-centric REST API operations. You can generate business models with visual tools, command-line tools, or discover models from an existing data source.

LoopBack is a high-performance Node.js interaction-tier framework for APIs. You can use the developer toolkit to generate Node.js APIs. You can use the API Manager or API Designer visual tools to create OpenAPI definitions.

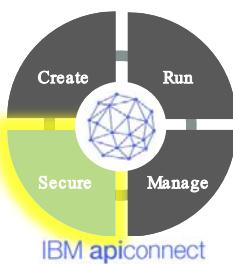
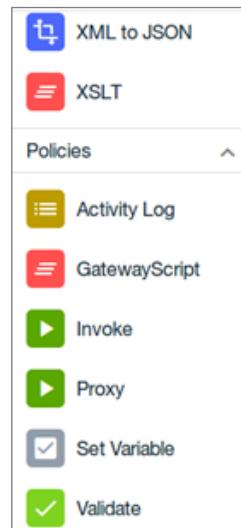
The LoopBack framework and API developer toolkit can be installed on the developer workstation for local development of LoopBack APIs. These components can also be installed when API Connect is installed.

Applications that are built with LoopBack can run natively on a Node runtime, or deployed to a Docker image that can be managed by the Kubernetes orchestration software.



Secure and control APIs

- Purpose-built, secure, and scalable gateway to enforce API policies at run time
- Comprehensive set of built-in security, traffic management, and mediation policies
- Ability to define custom user-defined policies with GatewayScript



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-8. Secure and control APIs

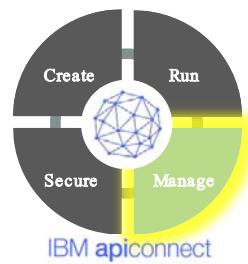
The API gateway enforces a set of security and message processing policies that you defined in the API gateway. You can choose the gateway type that meets your requirements. For example, the DataPower API Gateway generates and validates JSON Web Tokens, WS-Security metadata, and OAuth 2.0 authorization tokens.

The gateway also manages call quota and rate limits. The gateway also handles content-based routing, and transformation between message formats and transport protocols.



Manage APIs

- Manage API, Product, plans, versions, users, and subscribers
 - Organize APIs into catalogs and spaces
 - Manage API lifecycle
- Share and socialize APIs on self-service Developer Portal with API consumers
- Analyze API usage data to gain visibility and insight



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-9. Manage APIs

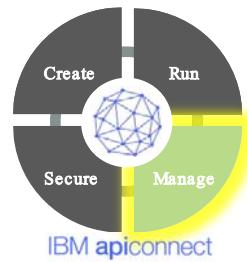
The API Management server manages the configuration and metadata on your published APIs.

Organize APIs into catalogs and spaces.

Review API versions and lifecycle events through the API Manager web interface. Subscribe to API lifecycle events.

Socialize APIs

- Enable self-service, company-branded Developer Portal for your API consumers
 - Built on robust, open source Drupal content-management system to customize developer experience
- Enable different API provider lines of business within an organization to socialize APIs to a single corporate Developer Portal
- API consumers can browse the available APIs, view API definitions, and test APIs
- Application developers can register applications, generate keys, and view analytics on usage
- Social portal with blogs, forums, and ratings



[IBM API Connect V2018 overview](#)

© Copyright IBM Corporation 2020

Figure 1-10. Socialize APIs

You socialize your APIs by using a Developer Portal. Socializing the APIs means that the APIs can be browsed and tested on the Developer Portal. For your application developers, the Developer Portal provides a highly customizable website to review, subscribe, and test APIs.

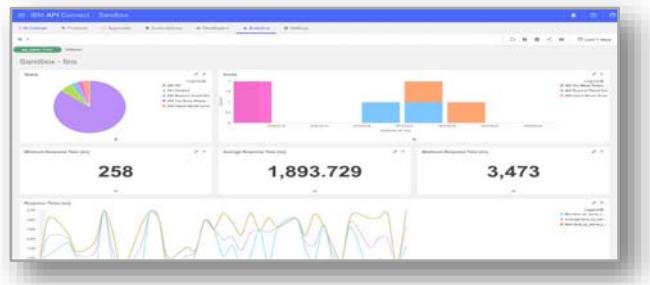
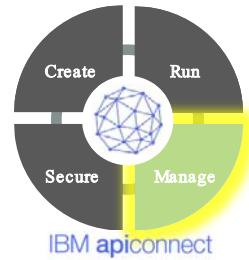
The Developer Portal is based on Drupal, which is a free and open source platform that you can use to manage the content of your website. The Developer Portal that is part of the API Connect software includes blogs, forums, and ratings for APIs as part of the standard offering.

Developers can register their client applications without interaction from the system administrator. Developers can use a set of developer community tools to share information. Application developers can browse available APIs, view the API details, and test API operations. Application developers can also register applications, generate API keys, and view analytics on API usage.



Analyze APIs

- Analyze API runtime usage data to gain insights and visibility
 - Powered by open source Elastic Stack
 - Understand performance with visualizations of call volume, error rates, and response times
 - Create custom dashboards and visualizations
- Analytics for both API provider and API consumer
- Out-of-the box dashboards and visualizations for common metrics



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

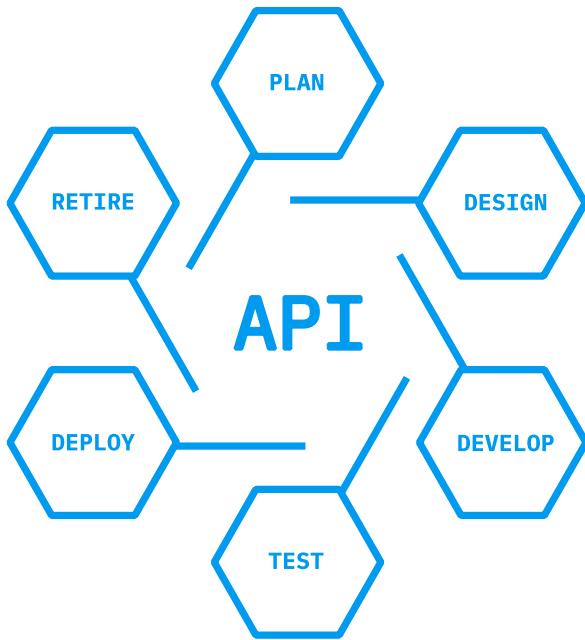
Figure 1-11. Analyze APIs

The analytics that are provided with API Connect provide you with visibility into API usage.

Retrieve API analytics in a dashboard view, based on the Elastic Stack open source project.

API providers can retrieve API analytics in a customizable dashboard view that displays visualizations that include API call volumes, error rates, and response time. Provider analytics can be viewed from the API Manager user interface. Consumer API analytics can be viewed from the Developer Portal.

Manage APIs through the API lifecycle



- Define and import REST or SOAP services
- Package APIs into products and tailor them for specific purposes
- Publish and promote products and APIs across different catalogs
- Manage the API lifecycle and version control from staging to retirement of APIs
- API subscription and application access control

[IBM API Connect V2018 overview](#)

© Copyright IBM Corporation 2020

Figure 1-12. Manage APIs through the API lifecycle

With IBM API Connect, you can manage APIs through the API lifecycle.

Define and import REST or SOAP services so that customers can evolve their existing corporate assets.

Package APIs into product groupings to target specific development teams or consumer organizations.

Publish and promote products and APIs across different catalogs to align with DevOps practices.

API lifecycle and version management from the initial staging of the APIs to deprecation and retirement to meet corporate governance needs.

API subscription and application access control. Corporations need to control the access to the APIs and impose rate limits on API use.

What is the API Connect Cloud?

- The **API Connect Cloud** is a collection of services that make up an API Connect installation, including configuration information and metadata



The **API Management service** maintains the runtime configuration of the API Connect Cloud: the servers, the API, the plans, and products



- The **API Connect Toolkit** is a set of development tools that run on the API developer's workstation
- It can stage, publish, and test APIs in the API Connect Cloud



- The **API Gateway** secures runtime access to APIs
- It enforces a set of message processing policies against API requests



- The **Developer Portal** is a repository for published API definitions, plans, and products
- Application developers register apps that use APIs on the portal

Figure 1-13. What is the API Connect Cloud?

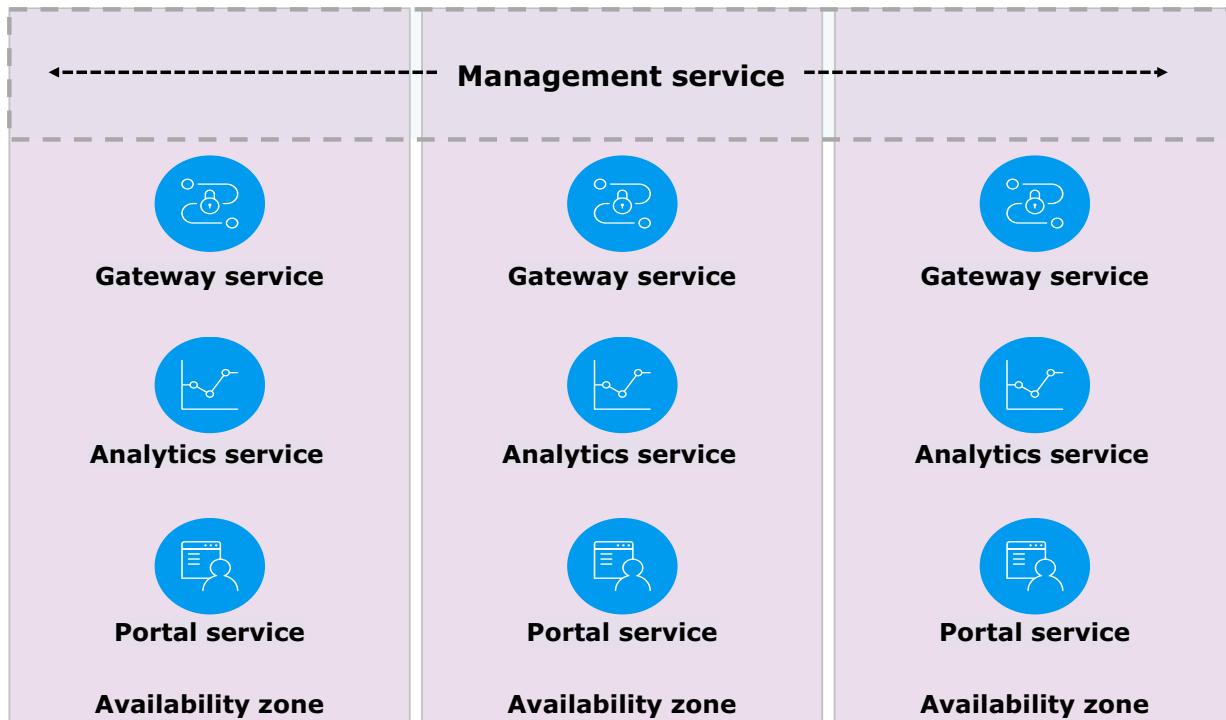
The **API Connect Cloud** is a collection of services that make up an API Connect installation, including configuration information and metadata.

- The **API Management service** maintains the runtime configuration of the API Connect Cloud: the servers, the API, the plans, and products.
- The **API Gateway** secures runtime access to APIs. It enforces a set of message processing policies against API requests.
- The **API Connect Toolkit** is a set of development tools that run on the API Developer's workstation. It can stage, publish, and test APIs in the API Connect Cloud.
- The **Developer Portal** is a repository for published API definitions, plans, and products. Application developers register apps that use APIs on the portal.
- The **Analytics Service** (not shown) provides visibility into API usage.

IBM Training



Cloud topology



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-14. Cloud topology

The Cloud topology consists of availability zones that contain the API Connect services - management, gateway, analytics, and portal services. An availability zone is a logical or physical set of data centers that contain one or more API Connect services.

Availability zones provide redundancy and failover if there are network issues.

Availability zones can contain one or more gateway services, analytics service, and portal service, but there is one management service that spans all availability zones.

A default availability zone is created during installation that includes a Management service.

An availability zone is a logical or physical set of data centers that contain one or more API Connect services. Availability zones provide redundancy and failover when network issues occur.

The management service is the control point. Within each availability zone, you can configure extra components for scalability:

1 –n gateway services

0 –n portal services

0 –n analytics services.

In this course, the on-premises cloud consists of a single availability zone that includes a management service. A gateway service, analytics service, and portal service were registered after product installation by using the Cloud Manager user interface.



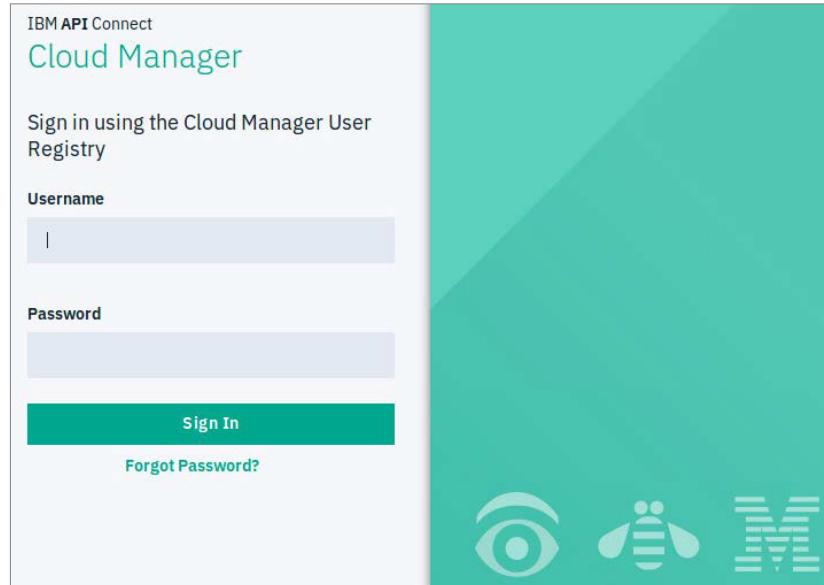
Information

The Cloud Manager topology consists of availability zones that contain the API Connect services, namely, the management, gateway, analytics, and portal services.

You review the services that are defined in Cloud Manager in the exercise that is named “Reviewing the course development and runtime environment”.

Defining the cloud topology: Cloud Manager

- Cloud Manager user interface enables a cloud administrator to define and manage the API Connect on-premises cloud



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-15. Defining the cloud topology: Cloud Manager

You configure and manage the servers that comprise your IBM API Connect on-premises cloud by using the Cloud Manager user interface.

The first time that you access the Cloud Manager user interface, you are prompted to change the cloud administrator password and address. The initial password change is already done for you in the course exercises.

The default user registry for Cloud Manager is an API Connect internal registry.

This registry holds the administrator account and any other users that you define to manage the on-premises cloud.

Cloud Manager tasks

You can use the Cloud Manager to define the API Connect Cloud when you perform functions:

- Create Provider organizations and invite users to serve as the owner
- Create and manage user roles and role defaults
- Create availability zones for services
- Register the relevant gateway, analytics, and portal services
- Associate an analytics service with a gateway to enable reports for API Events
- Configure resources for user authentication and TLS security
- Configure the connection to an existing SMTP mail server for system-generated emails
- Set the default gateway service for catalogs

Figure 1-16. Cloud Manager tasks

You can use the Cloud Manager to define the API Connect cloud with these functions:

- Create Provider organizations and invite users to serve as the owner
- Create and manage user roles and role defaults
- Create availability zones for services
- Register the relevant gateway, analytics, and portal services within availability zones to securely create, publish, and track APIs.
- Associate an analytics service with a gateway to enable reports for API Events
- Configure resources for user authentication, TLS security, and OAuth providers and make the resources visible to all or selected provider organizations
- Connect to an existing SMTP mail server and edit templates for system-generated emails
- Set the default gateway service for catalogs

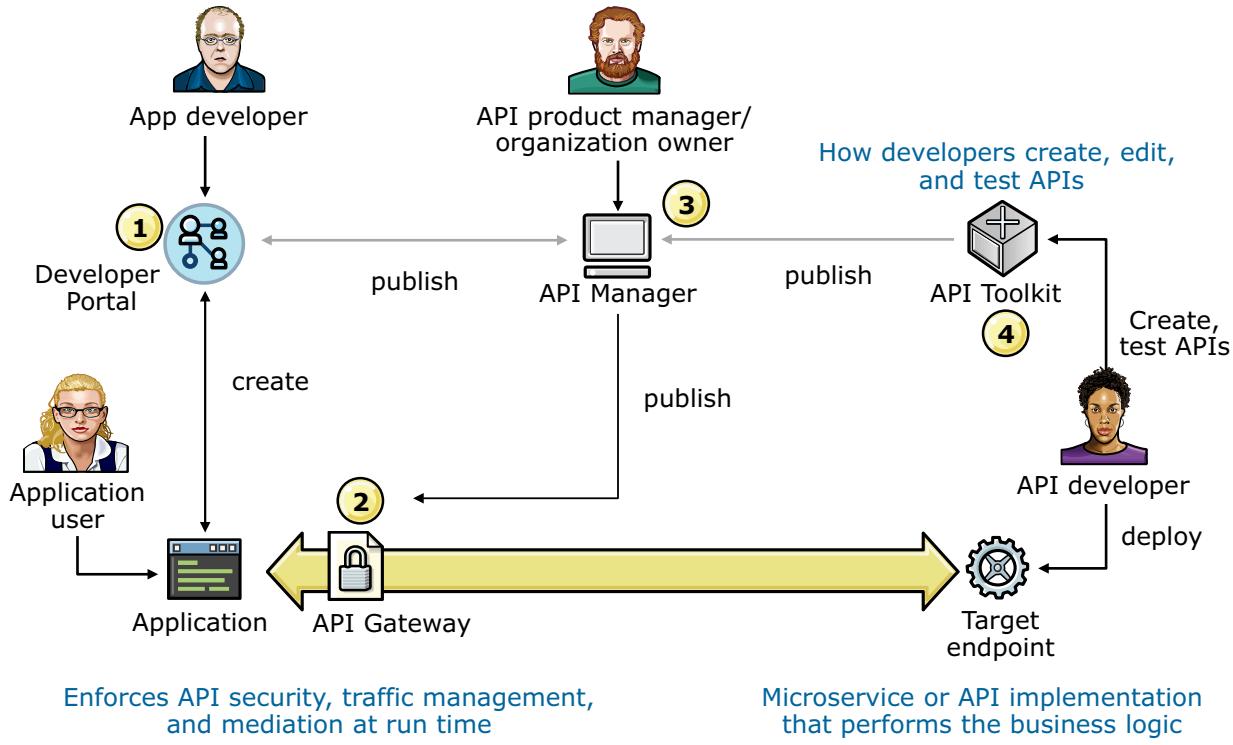


Information

These Cloud Manager administration tasks are already done on the student image where the exercises are run.

You get to review the configuration for the on-premises cloud in the first exercise.

API Connect: Other user interfaces and user roles



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-17. API Connect: Other user interfaces and user roles

The components of the API solution are as follows:

1. The Developer Portal is used to socialize the APIs, register applications, and subscribe to plans.
2. The API Gateway enforces API security, traffic management, and mediation at run time.
3. The API Manager controls the management and publishing of APIs, plans, and products.
4. The API Developer Toolkit is used to create, edit, and test APIs

The providers and the consumers of APIs can be categorized into different roles.

- An API or software developer is in the organization responsible for implementing the API operations. The API developer uses the **API Connect Toolkit** to develop, test, and publish APIs, applications, plans, and products.
- The API product manager or organization owner is the owner of a provider organization. This role is responsible for the review and approval of lifecycle changes with the API Manager web user interface.
- The application developer (or app developer) builds an application that uses published APIs. The app developer subscribes to APIs in the **Developer Portal**.

- The application user uses the application and its associated APIs that the app developer creates.

1.2. Gateway options

Gateway options

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-18. Gateway options

Topics

- API Connect overview
- Gateway options
- Software and hardware prerequisites

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-19. Topics

Security and control with industry-leading DataPower API gateway

- Signed and encrypted gateway image without external software dependencies to minimize risk
- Well-established API security policies to protect services and data across multi-clouds
- Scalable architecture to help meet high-availability workloads
- Optimized built-in policies for security, traffic management, and mediation
- Workload tenant isolation to optimize governance on a single appliance across multiple lines of business

Secure, scalable, and optimized



IBM DataPower Gateway



IBM DataPower Gateway Virtual Edition

[IBM API Connect V2018 overview](#)

© Copyright IBM Corporation 2020

Figure 1-20. Security and control with industry-leading DataPower API gateway

The IBM DataPower Gateway is a proven, industry-leading gateway for secure and controlled access to APIs.

The DataPower gateway is available as a physical or virtual appliance, Linux-based software, a Docker image, or as a cloud form factor.

DataPower in the cloud

The DataPower gateway is available in cloud, physical, virtual appliance, Linux, and Docker form factors

When DataPower runs in the cloud, it is running under a hypervisor, in Linux, or in a Docker container

- DataPower in a Docker container is supported in:
 - IBM Cloud
 - Amazon Web Services (AWS)
 - Google Cloud
 - Microsoft Azure

Figure 1-21. *DataPower in the cloud*

The DataPower gateway is available in multiple form factors, including cloud, physical or virtual appliance, Linux-based, and as a Docker image.

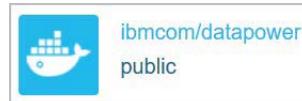
All the DataPower form factors provide the same level of robustness and scalability.

The Docker form factor has some limitations that are listed in the next slide.

The DataPower Gateway for Developers on Docker is targeted to developers that want to perform local testing with a DataPower gateway.

Downloading DataPower Gateway for Developers on Docker

- Docker
 - <https://hub.docker.com>
 - Search on “ibmcom/datapower”
 - Use this official IBM version



- IBM Fix Central (<https://www.ibm.com/support/fixcentral/>)
 - The download file (tar.gz) is under the firmware
 - Example: IDG-docker-2018.4.1.3
 - *idg_docker2018413.tar.gz*

fix pack: IDG-docker-2018.4.1.3-LTS-Fix-Pack

DataPower-2018.4.1.3-LTS-IDG-docker

The following files implement this fix.

 [idg_dk2018413.lts.tar.gz \(282.07 MB\)](#)

Figure 1-22. Downloading DataPower Gateway for Developers on Docker

To run the DataPower Gateway in a Docker container, run the Docker image that contains the DataPower Gateway.

The image requires a minimum of 4 GB RAM and 2 CPUs.

DataPower Gateway for Developers runs on Docker Engine V1.12 or later.

Certain functions are disabled or follow the Docker conventions when the DataPower Gateway runs in a Docker container.

- You cannot configure Ethernet interfaces, VLAN interfaces link aggregation interfaces, and network settings. All commands are disabled.
- You cannot configure the NTP service. All commands are disabled.
- You cannot configure a Secure Gateway client. All commands are disabled.
- You cannot set the date or time. The clock command is disabled.
- You cannot use services that are configured on other DataPower domains.



Information

When the DataPower Gateway is installed in a Kubernetes environment, such as the one that the course image uses, you can deploy the Docker-based dynamic gateway or an external non-Docker gateway. This course uses an external non-Docker IDG gateway that runs as a separate virtual appliance.

API gateway types

IBM API Connect provides two gateway types, DataPower API Gateway and DataPower Gateway (v5 compatible)

- **DataPower API Gateway**
 - The DataPower API Gateway is a new gateway that is designed with APIs in mind, and with the same security focus as DataPower Gateway (v5 compatible)
 - API Gateway is built specifically for handling APIs, with resulting performance benefits
 - Consider that use this gateway if you are running applications in a public or private cloud and want to use them as APIs
- **DataPower Gateway (v5 compatible)**
 - Available with IBM API Connect for a number of years
 - Broad range of policies
 - More flexible when custom policies are required
 - Consider using this gateway if you are an existing DataPower user and want to use your DataPower resources and knowledge

Figure 1-23. API gateway types

IBM API Connect provides two gateway types, DataPower API Gateway and DataPower Gateway (v5 compatible).

DataPower API Gateway:

- The DataPower API Gateway is a new gateway that is designed with APIs in mind, and with the same security focus as DataPower Gateway (v5 compatible).
- The DataPower API Gateway was built and optimized for the cloud. Consider that use this gateway if you are running applications in a public or private cloud and want to use them as APIs.

DataPower Gateway (v5 compatible):

- DataPower Gateway (v5 compatible) has been available with IBM API Connect for a number of years. This gateway provides a broad range of policies that are built in to the API Connect API assembly, including transformations, security policies, logic, and GatewayScript.
- Consider that use this gateway if you need custom policies or you have complex API assembly requirements.

For more information, see the Gateway comparison table in the IBM Knowledge Center for IBM API Connect at

https://www.ibm.com/support/knowledgecenter/SSMNED_2018/com.ibm.apic.overview.doc/r_apic_gateway_types.html



Information

The DataPower API Gateway is used in the course exercises.

1.3. Software and hardware prerequisites

Software and hardware prerequisites

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-24. Software and hardware prerequisites

Topics

- API Connect overview
- Gateway options
-  Software and hardware prerequisites

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-25. Topics



API Connect: Requirements

- Review the most recent software and hardware requirements:
 - <https://www.ibm.com/software/reports/compatibility/clarity/softwareReqsForProduct.html>

Continuous Delivery Product - Long Term Support Release ⓘ

IBM API Connect 2018.4.1.0

Detailed System Requirements

Report filters

Available Reports: 2018.4.1.3 maintenance level

Utilities: Regenerate Anytime, Download PDF, Print

Notes: Data as of 2019-03-11 03:04:27 EDT

Operating Systems: Hypervisors, Prerequisites, Supported Software, Hardware, Packaging List, Integrated Products

Show notes | Hide notes

Linux, Mac OS, Windows

Linux

Operating System	Operating System Minimum ⓘ	Hardware	Software	Components	Notes
Red Hat Enterprise Linux (RHEL) Server 7	7.4	x86-64	32, 64-Exploit, 64-Tolerate	Developer Toolkit -- CLI, InstallAssist	(2)
Ubuntu 16.04 LTS	Base	x86-64	32, 64-Exploit, 64-Tolerate		(2)
Ubuntu 18.04 LTS	Base	x86-64	32, 64-Exploit, 64-Tolerate		(2)

Hover over the components to view which parts of IBM API Connect depends on a particular operating system, hardware, or software level

Desktop Component Support: Developer Toolkit -- CLI, InstallAssist

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-26. API Connect: Requirements

Review the most recent software and hardware requirements from the IBM Knowledge Center for API Connect.

Type API Connect in the search field. Then, specify the version number. In the detailed system requirements page, hover over the component icons. Review the operating system, hardware, and software levels for a particular server component.

Unit summary

- Explain what is IBM API Connect
- Describe the key capabilities of API Connect
- Describe how API Connect manages API through the entire API lifecycle
- Identify the components of an API Connect on-premises cloud
- Describe the use of the Cloud Manager user interface to administer the cloud topology and resources
- Describe the different gateway types for securing and managing APIs
- Identify the requirements for installing an API Connect on-premises cloud

Review questions

1. True or False: API Connect enables both the creation of APIs and the full lifecycle management of APIs.
2. What is the role of the API gateway?
 - A. It secures API endpoints
 - B. It manages and monitors API traffic
 - C. It transforms API requests and responses
 - D. All of the above
3. Which capability can you find in the DataPower Gateway (v5 compatible) but not the DataPower API Gateway?
 - A. GatewayScript support
 - B. Rate limiting
 - C. Message transformation
 - D. Custom user-defined policies



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-28. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: API Connect enables both the creation of APIs and the full lifecycle management of APIs.

The answer is True.



2. What is the role of the API gateway?

- A. It secures API endpoints
- B. It manages and monitors API traffic
- C. It transforms API requests and responses
- D. All of the above

The answer is D.

3. Which capability can you find in the DataPower Gateway (v5 compatible) but not the DataPower API Gateway?

- A. GatewayScript support
- B. Rate limiting
- C. Message transformation
- D. Custom user-defined policies

The answer is D.

Exercise: Review the API Connect development and runtime environment

IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-30. Exercise: Review the API Connect development and runtime environment

Exercise objectives

- Test the operation of the private DNS on the image
- Review the Kubernetes runtime components
- Ensure that the API Connect pods are operational
- Sign on to the Cloud Manager graphical interface
- Review the provider organization in Cloud Manager
- Review the settings in Cloud Manager



IBM API Connect V2018 overview

© Copyright IBM Corporation 2020

Figure 1-31. Exercise objectives

Unit 2. API Connect development platform

Estimated time

00:30

Overview

This unit describes the purpose of the API Connect Developer Toolkit to create APIs and applications in the LoopBack framework. Learn the different approaches to creating API definitions and API applications.

How you will check your progress

- Review questions

Unit objectives

- Explain the purpose of the API Connect Developer Toolkit
- Identify the different installation options for the API Connect Developer Toolkit
- Identify the features of the apic lb command line utility
- Describe how to create APIs in API Connect

What is the IBM API Connect developer toolkit?

- You create API definitions and LoopBack applications with the **IBM API Connect developer toolkit**
- The **toolkit** is a set of applications that you extract to your workstation:
 - You define API interfaces with the API Designer web application
 - You generate LoopBack application components with the command line interface
 - The toolkit provides a command-line tool, **apic** that you can use to perform all API Connect tasks
- The toolkit is provided as executable files
 - Download the required compressed file and extract the contents
 - Available for MacOS, Linux, and Windows operating systems

Figure 2-2. What is the IBM API Connect developer toolkit?

You create API definitions and LoopBack applications with the IBM API Connect developer toolkit.

The toolkit provides a command-line tool, **apic** that you can use to perform all API Connect tasks.

The toolkit is provided as executable files, so no actual installation is necessary, you just need to download the required compressed file and extract the contents.

API Connect toolkit options

There are two toolkit options available:

- **CLI**
 - Provides a command line interface for working with API Connect
- **CLI + LoopBack + Designer**
 - A command line interface for working with IBM API Connect, including LoopBack support, and the API Designer user interface
 - You generate and configure Loopback applications with the ***apic lb*** command-line utility

Figure 2-3. API Connect toolkit options

The API Connect developer toolkit is available in two options:

- CLI: provides a command line interface for working with IBM API Connect.
- CLI + LoopBack + Designer: provides a command line interface for working with IBM API Connect, including LoopBack support, and the API Designer graphical interface.

In this course, you use the **apic lb** command to generate and configure LoopBack Node applications.

Toolkit prerequisites

- Before you install the API Connect product and the API Connect developer toolkit, ensure that you have all the prerequisite software installed
- The steps to install the API Connect toolkit are:
 1. Install and configure the prerequisite software for your operating system
 2. Install the Node runtime environment and `npm`
 3. Extract the toolkit option that you intend to use
 4. Verify the toolkit installation

[API Connect development platform](#)

© Copyright IBM Corporation 2020

Figure 2-4. Toolkit prerequisites

Before you install the API Connect product and the API Connect developer toolkit, ensure that you have all the prerequisite software installed. Confirm that you have the correct version of Node.js and NPM software and other operating system-specific prerequisite software on your workstation before you attempt to install the API Connect toolkit.

See the IBM Knowledge Center for API Connect for the prerequisite compilers and languages.

Review the supported level for Node.js

- In the IBM Knowledge Center for API Connect, find the version requirements in the table of contents
 - https://www.ibm.com/support/knowledgecenter/SSMNED_2018/com.ibm.apic.install.doc/overview_apimgmt_requirements.html
- Open the detailed system requirements for a product link. Then, generate the Software Product Compatibility Report
- In the Software Product Compatibility Report, review the **Compilers and Languages** section

Compilers and Languages			
Prerequisite	Version	Prerequisite Minimum	Components
Node.js	v8.0.0 and future fix packs	v8.0.0	 
	v10.0	v10.15.1	

IBM API Connect 2018.4.1
Detailed system requirements

API Connect development platform

© Copyright IBM Corporation 2020

Figure 2-5. Review the supported level for Node.js

The Software Product Compatibility report generates a list of prerequisite software, hardware, and operating system levels for an IBM product. In this example, you want to look up which Node.js runtime environment version API Connect toolkit requires. Open the IBM Knowledge Center and find the installation requirements topic. A link is provided where you can specify the product and version and generate the Software Product Compatibility Report.

Review the Node.js" version in the compilers and languages section of the prerequisites page. The IBM SDK for Node.js is a distribution of the Node.js runtime environment. The version numbers match the ones from the open source Node.js project.

Always refer to this report when you look up hardware and software levels. When IBM releases a fix pack, they update this report to match the requirements.

Verify the toolkit installation

- Run the “apic” command-line utility and verify the version number

```
$ apic version
```

- Start the LoopBack command-line interface (if this option of the toolkit is installed)

```
$ apic lb --help
```

Figure 2-6. Verify the toolkit installation

To verify the toolkit version number, run the apic command-line utility. Keep in mind that the `apic` command cannot check the version number on the Developer Portal, API Management, and API Gateway servers. You must check each one of these components individually.

When you install the API Connect developer toolkit, you must install the toolkit at the same version level as the other API Connect software.

To view the available LoopBack commands, type: `apic lb -help`



Information

The ‘apic lb’ command is a wrapper for the ‘loopback-cli’ module. Developers experience the same behavior when the ‘apic lb’ command is used as the ‘lb’ command. The LoopBack commands are described in the LoopBack documentation at loopback.io/doc

API Connect Toolkit: The apic lb command

- The API Connect V2018 toolkit **apic lb** command is used to create and scaffold LoopBack NodeJS applications
 - Create LoopBack applications and data sources
 - Generate a sample LoopBack application to implement APIs
 - Define LoopBack models, properties, and relationships

```
$ apic lb --help
Available commands:
apic lb acl
apic lb app
apic lb bluemix
apic lb boot-script
apic lb datasource
apic lb export-api-def
apic lb middleware
apic lb model
apic lb oracle
apic lb property
apic lb relation
apic lb remote-method
apic lb soap
apic lb swagger
apic lb zosconnectee
```

Figure 2-7. API Connect Toolkit: The apic lb command

The API Connect V2018 developer toolkit provides an option to install the command line interface with the LoopBack feature.

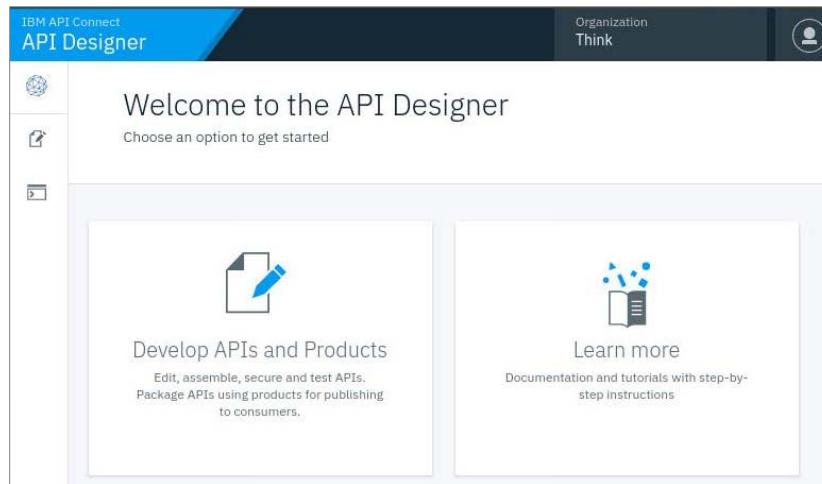
The apic lb command is used to create and scaffold LoopBack NodeJS applications.

With this command, you can create applications and data sources, generate sample LoopBack applications, and define models, properties, and relationships. The apic lb command also provides extensions to add IBM Cloud deployment artifacts to a LoopBack application.



API Designer

- Workstation-based graphical interface
- Packaged with the **CLI + LoopBack + Designer** toolkit option
- Develop APIs and products
- Subset of the function that is available with API Manager



[API Connect development platform](#)

© Copyright IBM Corporation 2020

Figure 2-8. API Designer

The API Designer is available when you extract the **CLI + LoopBack + Designer** toolkit option.

The API Designer is a graphical interface that is used to define API interfaces according to the OpenAPI specification.

The API Designer looks similar to the browser-based API Manager user interface. API Designer contains a subset of the function of the API Manager user interface.



Information

The API Manager is the graphical user interface that is used in this course to design API definitions and API policies.

The API Designer user interface is not used in this class.

How to create APIs in API Connect

- How do you define an **existing API** in IBM API Connect?
 - Define the API interface and operations by creating or importing an OpenAPI definition into API Manager
 - Publish the API definition to the API Gateway through API Manager
- How do you create an **API** in IBM API Connect?
 - Generate a LoopBack application with the API Connect toolkit
 - Create the models, properties, and relationships in the LoopBack application
 - Implement the business logic in Node.js
 - Deploy the Loopback application to a server
 - Generate the API interface and operations in an API definition
 - Import the API definition into API Manager
 - Publish the API definition to the API Gateway through API Manager

Figure 2-9. How to create APIs in API Connect

For a detailed discussion on how to call an existing API in API Connect, see the “Creating an API definition” unit.

To make an API available to client developers, you publish the API interface to the API gateway. In the unit named “Implementing APIs with the LoopBack framework”, and later units, you examine how to generate LoopBack scaffolding for applications and create APIs.

Unit summary

- Explain the purpose of the API Connect Developer Toolkit
- Identify the different installation options for the API Connect Developer Toolkit
- Identify the features of the apic lb command line utility
- Describe how to create APIs in API Connect

API Connect development platform

© Copyright IBM Corporation 2020

Figure 2-10. Unit summary

Review questions

1. True or False: You can install the API Connect toolkit either with or without LoopBack.
2. Which command starts the API Connect toolkit with LoopBack?
 - A. `lb`
 - B. `apic start lb`
 - C. `apic loopback`
 - D. `apic lb`
3. Which tasks can you perform with the “apic lb” command-line utility?
 - A. Generate an OpenAPI definition from an application
 - B. Create a data source
 - C. Create a Loopback sample application
 - D. All of the above



API Connect development platform

© Copyright IBM Corporation 2020

Figure 2-11. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: You can install the API Connect toolkit either with or without LoopBack.
The answer is True.

2. Which command starts the graphical API Designer application?
 - A. `lb`
 - B. `apic start lb`
 - C. `apic loopback`
 - D. `apic lb`
The answer is D.

3. Which tasks can you perform with the “apic lb” command-line utility?
 - A. Generate an OpenAPI definition from an application
 - B. Create a data source
 - C. Create a Loopback sample application
 - D. All of the above
The answer is D.



API Connect development platform

© Copyright IBM Corporation 2020

Figure 2-12. Review answers

Unit 3. Creating an API definition

Estimated time

01:00

Overview

This unit examines the API definition, a structured file that documents the API operations, parameters, and data types. You learn how to define your API interface. You also examine the role of the extensions that API Connect adds to the OpenAPI definition and how message processing policies are defined in the API assemble view.

How you will check your progress

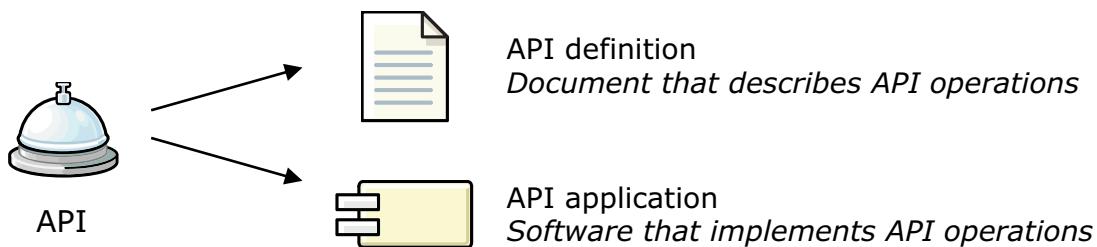
- Review questions
- Lab exercise

Unit objectives

- Explain the concept of an API definition
- Explain the purpose of the OpenAPI specification
- Describe how to create an API definition
- Describe the purpose of the target-url property
- Define an API operation
- Define query and path parameters
- Define request and response messages
- Describe the IBM API Connect extensions to the OpenAPI definition
- Describe the message processing policy assembly
- Identify the endpoint URL that gets called by the invoke message processing policy

What is an API definition?

- An **application programming interface** (API) defines business or technical capability as a set of operations
- An OpenAPI **API definition** is a standard, language-neutral way to specify the interface of a REST API
- Application developers can call API operations without seeing the API implementation, scanning API traffic, or relying on other documentation



[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-2. What is an API definition?

The OpenAPI **API definition** specifies the interface of a REST API in a standard, vendor-neutral, and language-neutral way. With a properly specified API definition, software developers and applications can discover and understand the capabilities of the service without access to the implementation source code, extra documentation, or inspecting network traffic.

IBM API Connect uses OpenAPI API definitions as the API interface format. You can import API definitions that other editors create, or export API definitions from API Connect toolkit.

Open API: REST API interface standard

- The **Open API** specification defines a standard format for declaring an REST API interface:
 - API metadata: Description and version
 - Operations
 - Data types
 - Parameters
 - Properties
- OpenAPI documents describe API operations and paths and are represented in either `YAML` or `JSON` formats
- When you create an **API** in API Connect, you declare an **OpenAPI definition**
 - IBM API Connect extends the Open API specification and adds message processing **policies** in the **assembly** section of the API definition



[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-3. Open API: REST API interface standard

The OpenAPI Specification (OAS) defines a standard interface description for REST APIs. The Open API 2.0 specification is a community-driven open specification within the OpenAPI Initiative, a Linux Foundation Collaborative Project. For more information, see:
<https://github.com/OAI/OpenAPI-Specification>.



Information

The definition files that are imported or created in this course conform to the OpenAPI Specification 2.0 that is identical to the Swagger 2.0 specification.

OpenAPI definition structure

- Swagger specification version (2.0)
- Info: API metadata
 - Name, description, version
- Schemes: Transfer protocol of the API. For example, https or https
- Host, base path
- Consumes and produces media types
- Paths
 - Operations
 - Request headers, parameters, and message body
 - Response headers, status code, and message body
- Definitions of schema data types
- Security definitions and schemes
- Security requirements
- Environment-specific properties
- Tags

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-4. OpenAPI definition structure

This page outlines the sections in an OpenAPI definition.

The **info section** provides metadata about the API that includes the name, description, and version information. These fields can be used by the test client in the API Developer Portal to display documentation from these fields.

The **host** and **base path** entries define the network endpoint for the API.

The API operations are defined in the **paths** section.

You define the structure of the request and response messages for each API operation. An operation is a combination of an HTTP method, such as GET or POST, with a resource path.

If you use a complex data type in an HTTP request or response message, you must define the schema type in the **definitions** section.

The **security schemes** explain the API authentication and authorization schemes.

The **security** requirements declare which security schemes apply to which operation.

The **properties** store values that are specific to a deployment environment. For example, you can store the target-url address for all the API operations. The target-url can be a combination of the host address, path, and search filters.

The **tags** entry stores keywords that make it easier for developers to find a list of APIs in a particular category.

The OpenAPI specification also supports vendor-specific elements in the definition document. These extension elements have an “x-” prefix in the name. For example, API Connect adds gateway processing policies, environment properties, and other metadata in the “x-ibm-configuration” entry.

IBM Training



Sample OpenAPI definition



```

notes.yaml (~/samples/notes) - gedit
Open ▾ Save
swagger: '2.0'
info:
  version: 1.0.0
  title: notes
basePath: /api
paths:
  /Notes:
    post:
      tags:
        - Note
      summary: Create a new instance of the model and persist it into the data source.
      operationId: Note.create
      parameters:
        - name: data
          in: body
          description: Model instance data
          required: false
          schema:
            description: Model instance data
            $ref: '#/definitions>Note'
      responses:
        '200':
          description: Request was successful
          schema:
            $ref: '#/definitions>Note'
      deprecated: false
YAML ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

```

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-5. Sample OpenAPI definition

The page displays a sample API definition file that is in YAML format. YAML is an acronym for “Yet Another Markup Language”. The output shows the source format of the OpenAPI definition file. When you import the file into a graphical API editor such as the API Manager, the editor creates a more easily readable graphical view that is called the Design View.



How do you create an API definition? (1 of 2)

- Sign in to API Manager. Then, select the “Develop APIs and Products” tile
- From the Develop page, click **Add**. Then, select **API**

A screenshot of the IBM API Connect API Manager interface. The top navigation bar includes "IBM API Connect" and "API Manager". On the right, there's a user icon and the text "Organization Think". The main content area has a title "Welcome to the API Manager" and a subtitle "Choose an option to get started". There are two main tiles: one titled "Develop APIs and Products" with a description "Edit, assemble, secure and test APIs. Package APIs using products for publishing to consumers.", and another titled "Manage Catalogs" with a description "Manage active APIs and consumers". The "Develop APIs and Products" tile is highlighted with a red rectangular border.

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-6. How do you create an API definition? (1 of 2)

API Connect provides web-based graphical environments for developing APIs and products.

Sign on to API Manager from a browser with your developer user account.

Click the “Develop APIs and Products” tile to open the Develop page in the browser.

From the Developer page, select the Add icon. Then, select API from the drop-down list.



How do you create an API definition? (2 of 2)

- To create an OpenAPI definition, select **Add > New API**
- To start with an empty OpenAPI definition, select **New OpenAPI**
- Provide a name, description, and version for the API
- Define the base path to differentiate the API from other APIs that you publish on the same gateway

The screenshot shows two main sections: 'Create' and 'Import'. The 'Create' section contains four options: 'From target service', 'From existing WSDL service (SOAP proxy)', 'From existing WSDL service (REST proxy)', and 'New OpenAPI'. The 'New OpenAPI' option is selected, indicated by a blue checkmark. The 'Import' section contains one option: 'Existing OpenAPI'.

Section	Option	Description
Create	From target service	Create a REST proxy that routes all traffic to a target API or service endpoint
	From existing WSDL service (SOAP proxy)	Create a SOAP proxy based upon a WSDL described target service
	From existing WSDL service (REST proxy)	Create a REST proxy based upon a WSDL described target service
	New OpenAPI	Compose a new REST proxy by defining paths and operations
Import		
Existing OpenAPI	Use an existing definition of a REST proxy	

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-7. How do you create an API definition? (2 of 2)

After you click the option to add an API, the page displays a number of options for creating the API. You can create a REST API definition to proxy to an existing REST service. You can create APIs from an existing WSDL service.

You can start with an empty OpenAPI definition document with the **New OpenAPI** option.

The **Existing OpenAPI** option imports an OpenAPI document from your workstation or a remote server.

You examine some of these options in other units and course exercises.

OpenAPI definition: Name, version, and licensing

- The entry provides metadata on the published API
- The **title** and **description** describe the name and a short overview of the API application
- The **version** lists the API implementation version
 - The OpenAPI definition does not define a version number scheme: The document stores the value as a string

Create New OpenAPI

Info	
Enter details of this API	
Title	savings
Name	savings
Version	1.0.0
Base path (optional)	
/savings	
Description (optional)	

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-8. OpenAPI definition: Name, version, and licensing

The beginning section of the API definition file lists metadata on the API.

On the first page of the “Create New OpenAPI”, the developer supplies the information that includes the title, name, version string, and an optional description.

The name and base path default to the same name as the title. These values can be overwritten.

Parts of an REST API definition

Type	Description	Example
Host	<ul style="list-style-type: none"> The name or address of the server that hosts the API In API Connect, the host is the address of the API Gateway 	https://apigw.think.ibm or Leave blank
Base path	<ul style="list-style-type: none"> The web route that identifies the API It appears immediately after the host name 	/savings
Consumes	The MIME media type of the HTTP request message	none
Produces	The MIME media type of the HTTP response message	application/json
Properties	Configuration settings that are applied to a specific deployment environment	target-url: http://savingsample.mybluemix.net/api/Plans

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-9. Parts of an REST API definition

This chart describes the functional parts of an OpenAPI definition file. The **host** entry lists the API server name or IP address. In API Connect, you can provide an environment property that the API gateway resolves at run time. If you omit this property, the host defaults to the API gateway address.

The **base path** entry displays the web route immediately after the API. The purpose of the base path is to separate the operations in this API from other APIs on the same server.

The **consumes** and **produces** entries explain how to interpret the data in the HTTP request message and response message. For example, a web form has a media type of text/html. If the API application returns a JSON object in the response message body, the message has a media type of application/json.

The OpenAPI definition file also stores environment-specific variables, which are known as **properties**. For example, the **target-url** can be specified for all the operations of the API. In the example, the target-url is specified as an actual URL address. Properties can be specified as API Connect context variables that start with a \$ sign.

NOTE: The **target-url** is the URL of the API service that is called. The operation paths are appended to the target-url to define the proxy endpoint.

It is important to distinguish between the proxy endpoint that is the back-end service that is called from the gateway. Client applications call the API at the gateway endpoint. The gateway routes or proxies the call to the proxy endpoint that is defined by the target-url.

Parts of an REST API operation

Type	Description	Example
Paths	The web resource, immediately after the base path	/estimate
Operation	The API operation is an action with an API resource	GET /estimate
Parameters	The input parameters of an API operation	deposit=300, rate=0.04 years=20
Responses	The HTTP status code and response message that are sent through the API implementation	200 OK {"balance":8933.42}
Definitions	The schema definition for the HTTP request or response message body	Property name: balance Type: float

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-10. Parts of an REST API operation

The REST API operation consists of five parts:

- The **path** is the name of the web resources that appears immediately after the base path, for example: /estimate. The path is appended to the target-url property, for example: <https://mybluemix.net/api/Plans/estimate>.
- The **operation** is the HTTP method that acts on the resource, for example: GET /estimate/.
- The **parameters** are the input parameters of an API operation, for example: deposit, rate, and years.
- The **responses** are the possible HTTP status codes and responses message that the API operation can return.
- The **definitions** are schema data types that appear in the HTTP request or response messages.

Review: HTTP methods in REST architecture

- **GET**
 - Retrieve information from a named resource on the server
 - The operation is safe and *idempotent*
- **POST**
 - Create or update information on a resource
 - The operation is *not idempotent*
- **PUT**
 - Store information at the named resource
 - The operation is *idempotent*
- **DELETE**
 - Remove information at the named resource
 - The resource does not have to be removed immediately: it is marked for deletion and no longer accessible

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-11. Review: HTTP methods in REST architecture

The most common HTTP methods in a REST architecture are: GET, POST, PUT, and DELETE. This slide quickly reviews the meaning of each operation in a REST architecture.

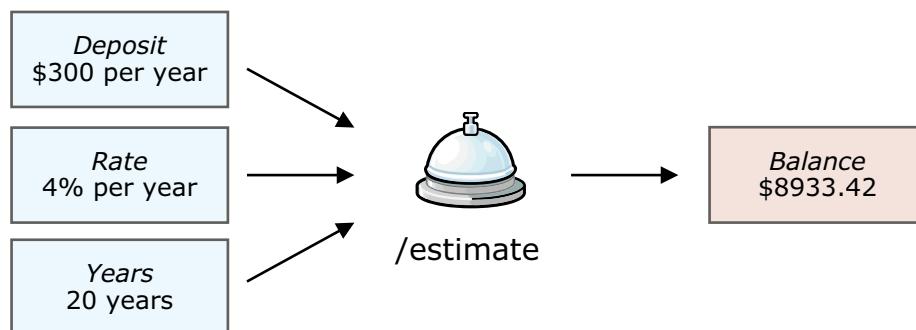
- A **GET** method retrieves the entity that is represented as a resource path on the server. The operation is safe because it is read-only: it does not change the information on the server. It is also *idempotent*: calling a GET method multiple times in succession returns the same value.
- The **POST** method represents an update to the entity on the server. Since POST methods change values, it is not a safe operation.
- The **PUT** method stores the entity in the request message onto the server.

One of the main points of confusion is whether to use a POST or PUT method to create or update a resource. The main difference between the two methods is that PUT is an *idempotent* operation, while POST is not *idempotent*. For example, you can use PUT to send a billing request. A PUT operation sends the entire billing record in the message body. Therefore, the entity on the server is the same every time you call PUT: it is an *idempotent* operation. However, this behavior is not ensured with a POST operation.

- The **DELETE** operation removes an entity on the server. The API implementation marks the resource as not available: the actual data record can be removed later.

Example: Savings plan estimate

- The GET /estimate operation calculates the savings account balance with compound interest
- This operation accepts three input parameters:
 - Deposit specifies how much to deposit in the account each year
 - Rate specifies the account interest rate
 - Years specify the number of years to calculate for the balance
- The operation returns the result in one output parameter:
 - Balance specifies the calculated account balance



[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-12. Example: Savings plan estimate

The **savings plan** REST API is a sample application that is hosted on the IBM Cloud architecture. The API calls the /estimate operation: a savings account with compound interest calculator. The operation takes three parameters: deposit, rate, and year. It returns the calculated balance in the response message.

In the example, the operation reads a deposit of 300 dollars at the end of each year. The interest rate is 4%, compounded annually. At the end of 20 years, the savings account balance is 8933.42 dollars.

Example: GET /estimate

- The GET /estimate operation accepts the input parameters as query parameters

```
GET /estimate?deposit=300&rate=0.04&years=20
Host: savingsample.mybluemix.net
Accept: application/json
```

HTTP request message

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{ "balance": 8933.42 }
```

HTTP response message

Figure 3-13. Example: GET /estimate

In this example, you send an HTTP GET request to the API service that is hosted on savingsample.mybluemix.net. The REST operation expects input parameters as query parameters: a set of key-value pairs after the question mark (?) character. The Content-Type response message header indicates that the application expects a response in the application/json format.

The savings plan API returns a response as a JSON object in the message body.

Step 1: Define a path, and operation

- The **Paths** represents a resource that is hosted on the server
 - For example, **/estimate**

API Setup	Paths
Security Definitions	NAME
Security	/estimate
Paths	

- An API operation consists of an **action** and a **path**
- In the REST architecture, each action has a specific meaning
 - For example, with the **GET /estimate** operation, you retrieve data on the savings plan estimate resource

Path name	/estimate
Operations	Add
NAME	
GET	⋮
POST	⋮

Figure 3-14. Step 1: Define a path, and operation

The next sequence of pages describe how to use the graphical features of API Manager to define the path and operations for an OpenAPI definition.

In this example, you want to define an OpenAPI definition that describes the GET /estimate operation in the IBM Cloud hosted sample application. In the first step, define the **path** in the API. The REST architecture is built on the concept of **resources**: a function or data record that you identify by a URL path. In this example, the **/estimate** path represents a savings plan estimate.

In the second step, you define an action, or a verb, that you associate with a path. The actions represent operations that you perform on the resource. The **GET /estimate** operation retrieves a savings plan estimate calculation.

Step 2: Define the result data type

In the **Definitions** section, you define the **custom data types** that operations use in the **request** and **response** messages

- In this example, the **savings-result** type defines a JSON object with one property: **balance**
- You specified this custom data type as the return value in the **GET /estimate** operation

The screenshot shows the 'Definitions' section of the API Manager interface. On the left, a sidebar lists 'API Setup', 'Security Definitions', 'Security Paths', and 'Definitions'. The 'Definitions' item is selected and highlighted in blue. The main area is titled 'Definitions' with the subtitle 'API request and response payload structures'. A large 'Add' button is at the top right. Below it is a table with columns 'NAME', 'TYPE', and 'DESCRIPTION'. A single row is present, showing 'savings-result' as the name, 'object' as the type, and a small bee icon. At the bottom of the table, it says 'No items found'. To the left of the table, there's a form for creating a new definition:

Name	savings-result										
Type	object										
Description (optional)											
Properties <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>PROPERTIES NAME</th> <th>PROPERTIES TYPE</th> <th>PROPERTIES EXAMPLE</th> <th>PROPERTIES DESCRIPTION</th> </tr> </thead> <tbody> <tr> <td>balance</td> <td>float</td> <td>8933.24</td> <td>Account balance</td> </tr> </tbody> </table>				PROPERTIES NAME	PROPERTIES TYPE	PROPERTIES EXAMPLE	PROPERTIES DESCRIPTION	balance	float	8933.24	Account balance
PROPERTIES NAME	PROPERTIES TYPE	PROPERTIES EXAMPLE	PROPERTIES DESCRIPTION								
balance	float	8933.24	Account balance								

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-15. Step 2: Define the result data type

In the definitions section of the Design view in API Manager, you define the API request and response message structure.

You then reference this schema definition in the response area of the GET /estimate operation.

Step 3: Define the input parameters and response message

- **Parameters** list the input fields, and **responses** list the output messages for the API operation

Parameters				
REQUIRED	NAME	LOCATED IN	TYPE	DESCRIPTION
<input checked="" type="checkbox"/>	deposit	query	float	
<input checked="" type="checkbox"/>	rate	query	float	
<input checked="" type="checkbox"/>	years	query	int 64	

Response		
STATUS CODE	SCHEMA	DESCRIPTION
200	savings-result	200 OK

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-16. Step 3: Define the input parameters and response message

This page shows you how to define the input parameters and response message in the API definition.

Type the input parameter name, location, description, and type in the **parameters** section of the API operation. When you mark a parameter as required, the API gateway checks that the parameter exists, and its value has the correct data type.

The **located in** field determines where the API operation expects to find the parameters:

- **Path:** A path segment in the URL
- **Query:** Query parameters in the URL
- **Header:** HTTP header
- **Form data:** HTML form in message body
- **Body:** Data in message body

- **Parameters** represent input data for the operation
- You can also define parameters at the path level
 - The OpenAPI specification defines data types in an API definition

- The **responses** list the expected status codes from the operation
 - If the operation completes successfully, it returns a status code of **200**, with the **savings-result** custom data type in the message body

Example: POST /estimate

- The POST /estimate operation accepts the input parameters as fields in a JSON object

```
POST /estimate
```

Host: savingsample.mybluemix.net

Accept: application/json

```
{ "deposit":300, "rate": 0.04, "years": 20 }
```

HTTP request message

```
HTTP/1.1 200 OK
```

Content-Type: application/json; charset=utf-8

```
{ "balance": 8933.42 }
```

HTTP response message

Figure 3-17. Example: POST /estimate

This example shows the expected request and response messages for calling the POST operation of the saving sample existing API. If you build the API definition from scratch, you need to know these values.

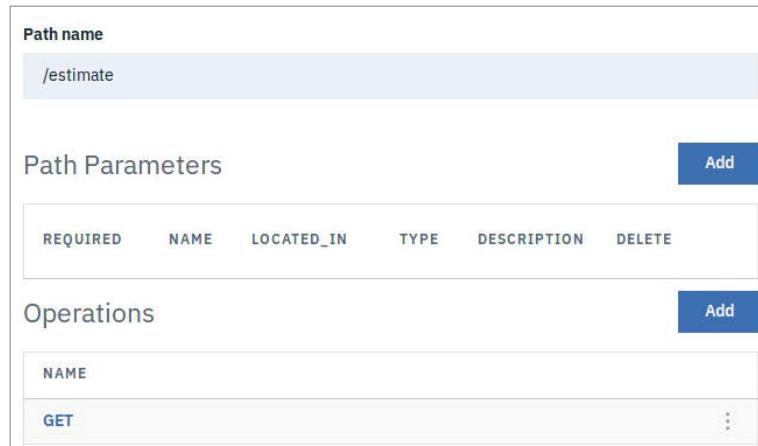
The **POST /estimate** operation accepts the input parameters as fields in a JSON object.

This version of the savings plan estimate function has the same behavior as the **GET /estimate** operation.

Instead of sending the parameters that are appended to the target-url, the parameters are located in the body of the request message.

Step 1: Create a POST operation in the existing path

From the existing
/estimate path, create a
second operation with the
POST method



REQUIRED	NAME	LOCATED_IN	TYPE	DESCRIPTION	DELETE

NAME		⋮
GET		

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-18. Step 1: Create a POST operation in the existing path

The next sequence of pages describe how to add a POST operation to the existing API path.

From the existing /estimate path, create a second operation with the POST method.

The POST method uses the request and response message layouts that are described on the previous page.



Step 2: Define the plan schema data type

Edit definitions

Name	plan
Type	object
Description (optional)	
Parameters for a savings plan calculation	

The **plan** schema type is a JSON object that holds the input parameters for the **POST /estimate** operation

- In this case, the **deposit**, **rate**, and **years** properties are fields in a JSON object
- At run time, the API caller sends a JSON object with these three fields in the HTTP message body

Properties			
PROPERTIES NAME	PROPERTIES TYPE	PROPERTIES EXAMPLE	PROPERTIES DESCRIPTION
deposit	float	300.00	Annual deposit
rate	float	0.04	Interest rate
years	integer	20	Years

Creating an API definition

© Copyright IBM Corporation 2020

Figure 3-19. Step 2: Define the plan schema data type

Define the schema type that holds the input parameters for the POST /estimate operation as a JSON object that contains three properties: deposit, rate, and years.

The API caller sends the plan JSON object with these three properties in the body of the HTTP request message.

Step 3: Define the input parameter in the message body

Parameters				
REQUIRED	NAME	LOCATED IN	TYPE	DESCRIPTION
<input checked="" type="checkbox"/>	plan	body	plan	

STATUS CODE	SCHEMA	DESCRIPTION
200	savings-result	200 OK

- In the **POST /estimate** operation, you replace the input parameter with a single custom data type named **plan**
- In the following step, you define the **plan** object with the three input parameters

Since both GET and POST operations return the result in the same manner, the **response** section in the operation also returns a message with a **savings-result** schema definition

Figure 3-20. Step 3: Define the input parameter in the message body

On this page, you see how to define the request parameters so they are passed in the body of the request message. The request parameters are defined by the plan schema definition that includes the deposit, rate, and years properties as a JSON-formatted object in the body of the request message.

Since both GET and POST operations return the result in the same manner, the response section in the operation also returns a message with a **savings-result** schema definition



API definition: Source view

Develop
savings 1.0.0

Design Source

```

1 swagger: '2.0'
2 info:
3   title: savings
4   x-ibm-name: savings
5   version: 1.0.0
6   schemes:
7     - https
8   basePath: /savings
9   produces:
10    - application/json
11   consumes:
12    - application/json
13   securityDefinitions:
14     clientIdHeader:
15       type: apiKey
16       in: header
17       name: X-IBM-Client-Id
18   x-ibm-configuration:
19     phase: realized
20     testable: true
21     enforced: true
22     properties:
23       target-url:
24         value: 'http://savingsample.mybluemix.net/api/Plans'
25         description: URL of the proxy policy
26         encoded: false
27     cors:
28       enabled: true

```

- You can review and edit the OpenAPI definition directly as a YAML text file in API Manager

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-21. API definition: Source view

The OpenAPI specification supports two API definition file formats: JavaScript object notation, or JSON, and Yet Another Markup Language, or YAML. IBM API Connect exports API definitions in the YAML file format.

The source view displays an OpenAPI definition in the raw text format. You can directly edit the text version of the OpenAPI definition in the API Manager, or any text editor.

Example: savings.yaml

```
swagger: '2.0'
info:
  title: savings
  x-ibm-name: savings
  version: 1.0.0
schemes:
  - https
basePath: /savings
produces: - application/json
consumes: - application/json
securityDefinitions:
  clientIdHeader:
    type: apiKey
    in: header
    name: X-IBM-Client-Id
x-ibm-configuration:
  phase: realized
  testable: true
  enforced: true
  properties:
    target-url:
      value: 'http://savingsample.mybluemix.net/api/Plans'
```

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-22. Example: savings.yaml

IBM API Connect also implements a number of extensions to the OpenAPI definition structure. For example, the **x-ibm-name** property stores the API name. If you export this OpenAPI definition file, the other platforms ignore extension properties by default.

IBM extensions to the OpenAPI definition format

- In addition to the properties in the OpenAPI specification, IBM API Connect defines a set of extensions to the API definition file
 - An extension property starts with the `x-ibm-` prefix in its name
 - Other platforms and tools safely ignore the IBM extension settings in an exported API definition file
- Examples of extension properties:
 - Setting to enable cross-origin resource sharing
 - Setting to enable API subscriptions
 - Setting to enable API testing
 - Message processing policies
 - Environment-specific properties

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-23. IBM extensions to the OpenAPI definition format

The OpenAPI specification defines the **API definition** as strictly an interface file: it describes the input and output messages for each operation in the API.

IBM API Connect extends the role of the OpenAPI definition file to several use cases. The API Gateway server hosts API operations according to the OpenAPI definition file. It also enforces a set of message processing rules that you define as an **assembly** extension. The API Manager reads the **lifecycle** extension to determine whether an API is ready for deployment to a staging or production environment. Last, the Developer Portal reads the **subscriptions** and **testing** extension to determine whether application developers can subscribe and test a published API.

In the next series of slides, you examine the role of these extension settings in detail.

Extensions: Lifecycle settings

API Setup	Lifecycle
Security Definitions	
Security	
Paths	
Definitions	
Properties	
Target Services	
Categories	
Lifecycle (optional)	
	Realized
	<input checked="" type="checkbox"/> Enforced
	<input checked="" type="checkbox"/> Testable
	<input checked="" type="checkbox"/> CORS
	<input type="checkbox"/> Application authentication
	<input type="checkbox"/> Buffering

- The **lifecycle** property saves the current development and deployment lifecycle maturity of the API
 - API Connect defines three phases: **identified**, **specified**, and **realized**
- The **enforced** property defines whether the API Gateway validates operation calls against the API definition
- The **testable** property determines whether the Developer Portal test client can test the API
- The **cross-origin resource sharing** determines how to interpret the access-control-allow-origin HTTP header

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-24. Extensions: Lifecycle settings

The purpose of the lifecycle extension property is to state the current development state of the API. In the **identified** state, the API definition and the API implementation are not complete. In the **specified** state, the API definition is complete, but the API application is not yet implemented. In the **realized** state, both the API definition and API implementation are complete.

The remaining options in the lifecycle section control other settings in the API Connect environment.

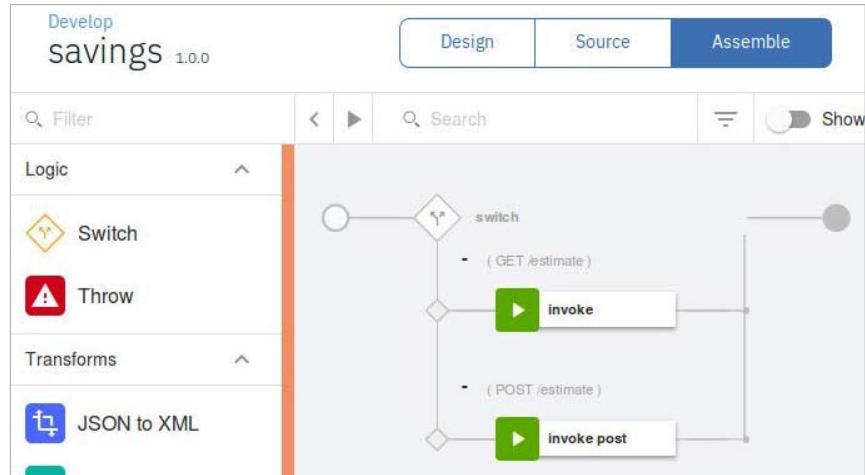
The **enforced** setting determines whether the API gateway enforces the settings in the API definition. If this setting is disabled, API is not exposed on the gateway.

The **testable** setting determines whether an application developer can review and test the API in the Developer Portal. If this setting is disabled, the operation does not appear in the test client.

The **CORS** setting determines whether the API supports the **Cross-Origin Resource Sharing** scheme. With the CORS scheme, a web server can use web resources on a named third-party server with a specific set of rules. For more information about this scheme, see: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS



Extensions: Message processing policy assembly



- In API Connect, you can define a set of message processing rules that apply to all operations in the API
 - At run time, the API Gateway enforces the policies that you define in the API assembly view
- You explore the concept of message processing policies in a later unit and exercise

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-25. Extensions: Message processing policy assembly

By default, the API gateway acts as a proxy: it receives and validates API operation requests, and it calls the actual API implementation that is hosted in your network. You can customize the steps before and after the API application invocation in the **assembly** view.

You explore the concept of message processing policies in a later unit and exercise.

Example: The invoke policy

- When you create an API definition, IBM API Connect defines an assembly with one message processing policy: an **invoke** operation



- In this example, the invoke operation forwards all API calls to the API implementation
 - The `$(target-url)` variable represents the **host** name of the server that hosts the API implementation
 - The `$(api.operation.path)` variable represents the path of the operation, for example: /estimate

[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-26. Example: The invoke policy

When you create an API definition, API Connect defines an assembly with a default invoke message processing policy.

In the properties window for the invoke policy, you can specify properties such as the URL to be invoked and the operation type.

Unit summary

- Explain the concept of an API definition
- Explain the purpose of the OpenAPI specification
- Describe how to create an API definition
- Describe the purpose of the target-url property
- Define an API operation
- Define query and path parameters
- Define request and response messages
- Describe the IBM API Connect extensions to the OpenAPI definition
- Describe the message processing policy assembly
- Identify the endpoint URL that gets called by the invoke message processing policy

Review questions



1. What is the `host` property?
 - A. It stores the API Gateway server name
 - B. It stores the API application server name
 - C. It stores the API Manager server name
 - D. It stores the API Connect Toolkit host name
2. What is the purpose of the `definitions` section?
 - A. It stores environment-specific values
 - B. It stores a list of published APIs in a catalog
 - C. It stores a list of HTTP response status codes
 - D. It stores type definitions for the message body
3. True or False: An `operation` combines a path with an HTTP verb, parameters, and definitions for requests and responses.

Creating an API definition

© Copyright IBM Corporation 2020

Figure 3-28. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers



1. What is the `host` property?
 - A. It stores the API Gateway server name
 - B. It stores the API application server name
 - C. It stores the API Manager server name
 - D. It stores the API Connect Toolkit host name

The answer is A
2. What is the purpose of the `definitions` section?
 - A. It stores environment-specific values
 - B. It stores a list of published APIs in a catalog
 - C. It stores HTTP response message parameters
 - D. It stores type definitions for message body data

The answer is D
3. True or False: An `operation` combines a path with an HTTP verb, parameters, and definitions for requests and responses.

The answer is True.

Exercise: Create an API definition from an existing API

Creating an API definition

© Copyright IBM Corporation 2020

Figure 3-30. Exercise: Create an API definition from an existing API

Exercise objectives

- Review an existing API service endpoint
- Create an API definition in API Manager
- Review the operations, properties, and schema definitions in an API definition
- Test the API GET operation in the assembly test facility
- Create a POST operation for the existing service endpoint
- Test the API POST operation in the assembly test facility



[Creating an API definition](#)

© Copyright IBM Corporation 2020

Figure 3-31. Exercise objectives

Unit 4. Defining APIs that call REST and SOAP services

Estimated time

01:00

Overview

With API Manager, you can quickly expose your existing cloud and enterprise services in the API gateway. This unit examines how to define API operations that call existing REST or SOAP APIs.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Explain the role of the API gateway in exposing existing services
- Explain how to expose an existing REST service in an API definition
- Explain how to expose an existing SOAP service in an API definition

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-1. Unit objectives

API runtime message flow

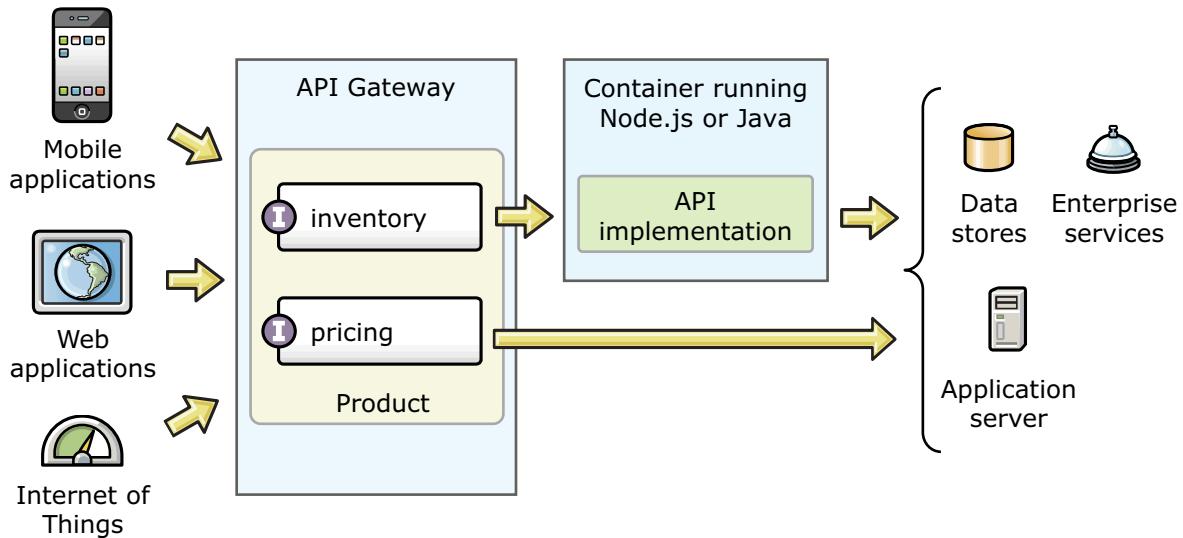


Figure 4-2. API runtime message flow

In this network diagram, you have a number of different client types that want to call your APIs. For **existing APIs** that already support a range of data types and clients, you can proxy these existing services in your API gateway. If you want to support new computing client types, or to quickly build a different interaction flow, you can build an **interaction API** that mediates and transforms messages to your system APIs.

A product is a collection of APIs that are organized together. You can publish many APIs in a product.

Scenario 1: Proxy existing APIs

- Your company already has a number of web services
 - Organizations within your company host web services to easily share information within the company
 - The business applications or business process management software that you use already hosts a number of services
- You can define and manage these **existing APIs** at your API gateway to make the service available beyond your company's network
- OpenAPI (Swagger 2.0) is used to create language-independent APIs that proxy to existing back-end implementations

Figure 4-3. Scenario 1: Proxy existing APIs

Your existing API implementations work well with the range of third-party, Business Partner, and internal applications. In this case, you might need to proxy these services in the API gateway only.

Scenario 2: Implement APIs

- Your company wants to design and develop APIs that keep up with the demands of your application developers and Business Partners
 - Whereas enterprise systems maintain stable, uniform standards throughout the company, the digital ecosystem must constantly add new features and standards that your customers demand
- You build APIs with your choice of language
 - **Node.js** is a modern, highly scalable programming model that efficiently handles requests with an event-driven architecture
 - **Java** is an established, programming model that is familiar to enterprise application developers
- The API Connect toolkit provides tools to build APIs with the LoopBack framework on Node.js
 - You can use your own development tools and processes to build interaction APIs on other language and platforms

Figure 4-4. Scenario 2: Implement APIs

One of the main drivers for building APIs is to support your customer demands. Your customers' demand that your APIs support the most recent computing standards, device types, and programming languages. Whereas enterprise systems maintain stable, uniform standards that change slowly, your digital system must constantly update itself – otherwise, your customers move their business to your competitors that support their needs.

With IBM API Connect, you have the choice to build your interaction APIs with the programming language of your choice. The API Connect Toolkit supports Node.js application development. You can also build APIs with other languages that your developers understand, such as Java and the call them from the API definition that is defined in API Connect.

What types of APIs can you define?

- REST API is a simple set of HTTP-based interactions, found in web applications
 - You map functions to web resource paths
 - Applications call API paths with HTTP methods, such as GET and POST
 - Applications exchange data with an API in JSON or XML
- SOAP API is an enterprise integration standard, is a specification for XML-based message exchange
 - You map functions to SOAP operations
 - Applications call API operations in an XML-based SOAP message format
 - Applications exchange data in XML

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

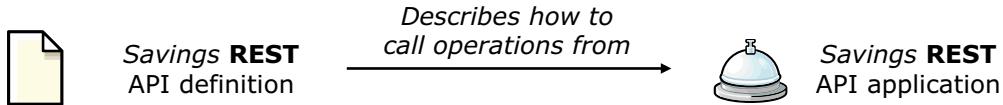
Figure 4-5. What types of APIs can you define?

You can define two types of APIs in the API Manager.

- The first type is a REST API: A set of operations based on simple HTTP interactions. In this style, functions map to web resource paths. Applications send an HTTP request to a web path to call API operations. The API and the application exchange data through JSON or XML messages.
- The second type is a SOAP API: An enterprise integration standard, found in service-oriented architecture (SOA) and enterprise service bus (ESB) infrastructure. In this style, functions map to SOAP operations. Applications specify the name of an API operation in an XML-based SOAP message format. The API and the application exchange data through XML data within the SOAP message.

Scenarios

1. Define a REST API that calls an existing REST application



2. Define a SOAP API that calls an existing SOAP application

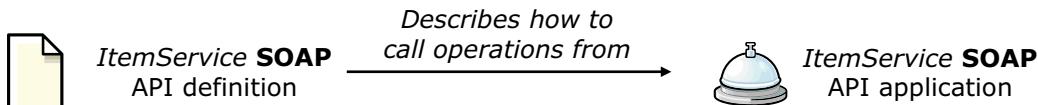


Figure 4-6. Scenarios

In a previous unit, you learned about the purpose and structure of the API definition document. As an API developer, you create an OpenAPI definition to describe the operations of an API.

This presentation takes a closer look at the scenarios in which you define an API. For an existing REST API, you can create an OpenAPI definition with the API Manager browser-based user interface.

Not all APIs follow the REST architecture. Traditional enterprise applications rely on the SOAP specification to remotely bind and invoke services. SOAP services rely on another standard to describe its interface: the Web Services Description Language (WSDL) document. In this second scenario, you can specify an OpenAPI definition based on an existing WSDL document.

Looking forward, you might want to modernize an existing SOAP service and provide a REST API. In this third scenario, you must specify an OpenAPI definition and policy assemblies to convert a call from REST to a SOAP request.

4.1. Proxy an existing REST API

Proxy an existing REST API

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-7. Proxy an existing REST API

Topics

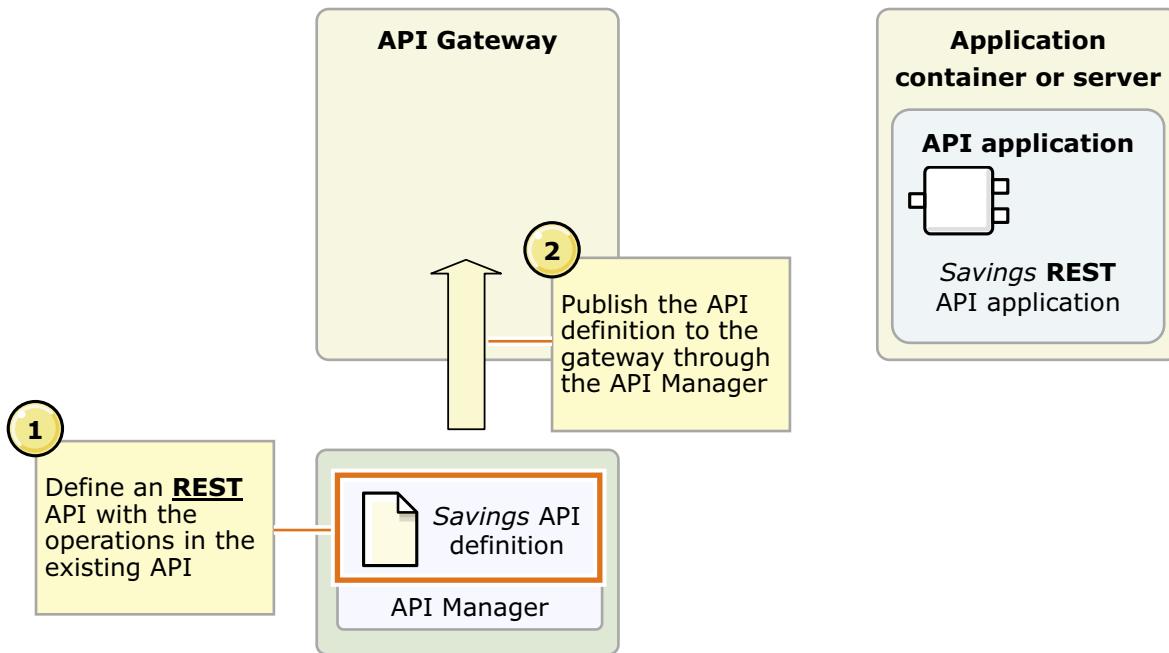
- ▶ Proxy an existing REST API
 - Proxy an existing SOAP API

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-8. Topics

Scenario one: Define a REST API to proxy to an existing REST service



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-9. Scenario one: Define a REST API to proxy to an existing REST service

In this scenario, you define an API that forwards, or proxies, requests to an existing REST API. The REST API can exist within your Cloud network, as an endpoint in an enterprise application, or as an API that is hosted outside your company.

In a previous lab exercise, you defined an OpenAPI document that describes the operation from the Savings REST API application. A REST API was created in the API editor from an existing REST service and the interface was saved as an OpenAPI definition. You published the API definition automatically when you tested the API in API Manager, which makes the service available on the API gateway.

Scenario one: Gateway forwards REST API requests

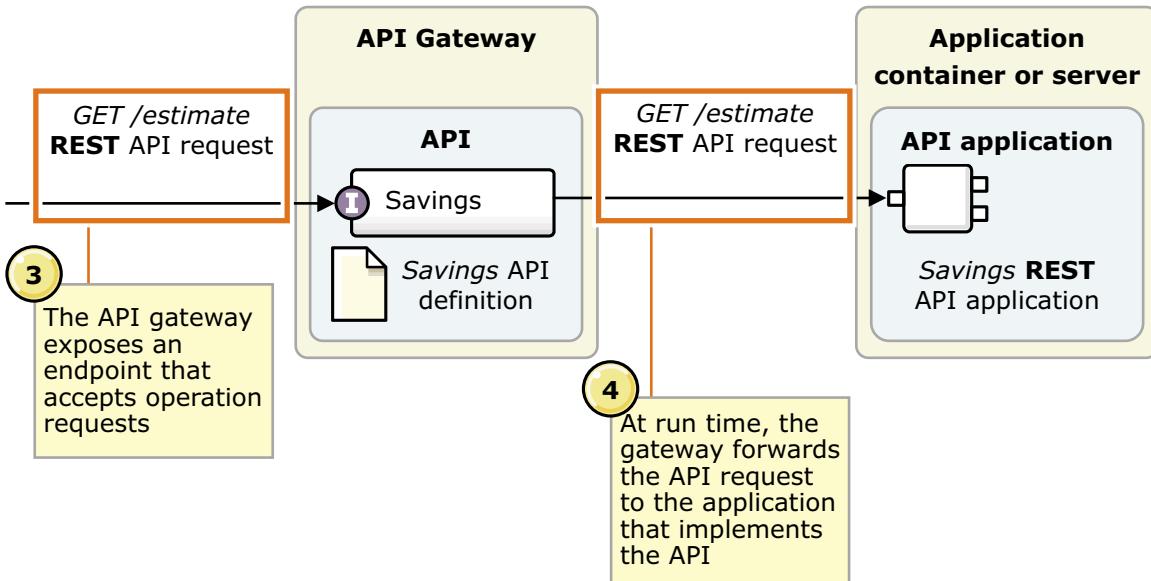


Figure 4-10. Scenario one: Gateway forwards REST API requests

When the application developer calls an operation from the Savings REST API, the API gateway intercepts the request. The gateway validates the API request against the interface in the OpenAPI definition. If the request is valid, the gateway forwards the request to the endpoint address where the application implements the savings API.

The architecture of API Connect divides an API implementation into two parts: the API gateway verifies and enforces the interface, while an application server or container hosts the application that implements the API operations.

In this example, the gateway forwards the `GET /estimate` API path and operation call to the implementation at the `savingsample.mybluemix.net` website.

Step 1: Create a REST API definition in API Manager

1. Start the API Manager web application
2. Select the Develop option
3. Click Add Then, select API
4. Select **From target service**
 - If the existing API implementation already has an OpenAPI definition, you can import it directly into API Manager

The screenshot shows the 'Add API' interface. Under the 'Create' heading, there are three options:

- From target service**: Selected. Description: Create a REST proxy that routes all traffic to a target API or service endpoint.
- From existing WSDL service (SOAP proxy)**: Description: Create a SOAP proxy based upon a WSDL described target service.
- From existing WSDL service (REST proxy)**: Description: Create a REST proxy based upon a WSDL described target service.

Below the options is a diagram illustrating the architecture:

```

graph LR
    App[App] <-->|OpenAPI| APIProxy((API Proxy))
    APIProxy <-->|Target Endpoint| Target[Target Endpoint]
  
```

The diagram shows an 'App' connected to an 'OpenAPI' document, which is then connected to an 'API Proxy' (represented by a green circle). The 'API Proxy' is also connected to a 'Target Endpoint' (represented by a server icon).

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-11. Step 1: Create a REST API definition in API Manager

To create an OpenAPI definition of a REST proxy to an existing target API, start the API Manager web application. Select the Develop APIs and products tile, or select the Develop option from the side menu. Click Add. Then, select API from the drop-down list. In the Add API page, select the option to create the API from a target service.



Note

API Manager provides options to create a new OpenAPI or to create APIs from existing REST and SOAP services. The from target service option makes it easy to create a proxy to an existing REST service. This example uses the same target service that you saw in an earlier presentation that describes the creation of an OpenAPI from scratch.

Step 2: Define the base path and target endpoint

The screenshot shows a configuration interface for defining an API. It includes fields for Version (1.0.0), Base path (optional) (/savings), and Description (optional). Below this, there's a section for Target Service URL with a placeholder 'Enter the URL for the target service you would like to proxy'. A specific URL is entered in the Target service URL field: <http://savingsample.mybluemix.net/api/Plans>. At the bottom are 'Cancel' and 'Next' buttons.

1. Type the title, name, and version number for the API definition
2. Define a base path to differentiate the API from other APIs that are hosted at the gateway
3. Specify the target endpoint for the API as the network location of the existing API implementation

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-12. Step 2: Define the base path and target endpoint

When you create an OpenAPI definition from scratch, you enter metadata on the API: the name, description, contact information, and version number for the API definition. The API gateway does not parse this information: it is kept for documentation purposes. One exception is the version number: you can configure API Manager to manage versions with API lifecycle management.

The **base path** uniquely identifies this API definition from other API definitions that are hosted at the gateway. Provide a unique path prefix in this document.

In the target service URL section, specify the **target endpoint** for the API definition. The **target** endpoint is the network location for the API application that implements the interface. In the lab exercise, the `savingsample.mybluemix.net` website hosts a copy of the API application. When you type a value in the target service URL, the value is placed in the **target-url** field of the YAML file that gets generated for the API.

Step 3: Edit the generated OpenAPI definition

- API Manager generates the basic OpenAPI definition with the values that you specified
- Click **Edit API** to complete the configuration of the API definition

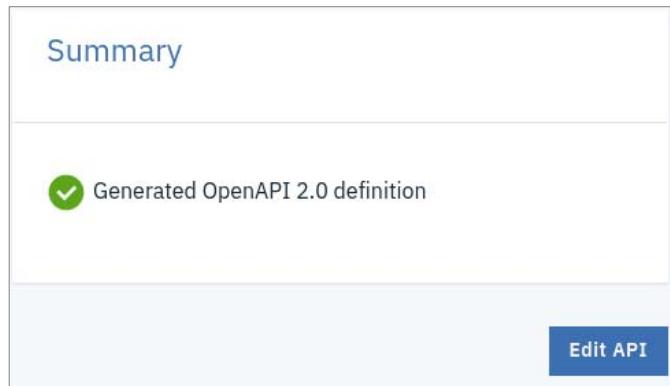


Figure 4-13. Step 3: Edit the generated OpenAPI definition

When you complete the prompts in the pages that follow the selection of the from target service option, API Manager generates the OpenAPI definition. Click **Edit API** to complete the configuration of the API definition.

Step 4: Select the gateway type

- Ensure that API Setup is selected in the Design view
- Scroll down to examine the options for the gateway type
- Change the gateway type to the preferred gateway that calls the API

The screenshot shows two overlapping screens. On the left, the 'Info' section of the 'API Setup' page for an API named 'savings' is displayed. The 'Title' field contains 'savings', 'Name' contains 'savings', and 'Version' is '1.0.0'. On the right, a separate window titled 'Gateway Type' is open, asking 'Select the gateway type for this API'. It contains two radio button options: 'DataPower Gateway (v5 compatible)' (unselected) and 'DataPower API Gateway' (selected).

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-14. Step 4: Select the gateway type

Ensure that API Setup is selected in the Design view.

Scroll down to examine the options for the gateway type.

Change the gateway type to the preferred gateway that calls the API.

For more information about gateway types, see the IBM API Connect V2018 Overview unit.

Step 5: Add definitions for response data type

- From the Definitions option
 - Click **Add**.
- Edit the definition and add the properties
 - Then, **Save**

PROPERTIES NAME	PROPERTIES TYPE	PROPERTIES EXAMPLE	PROPERTIES DESCRIPTION
balance	float	8933.42	Account balance

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-15. Step 5: Add definitions for response data type

Add the schema definition for the response message from the Definitions option of the Develop page in API Manager.

Specify the name of the definition and add the properties. Save when completed.



Step 6: Add a path and operations

- Create a path to call the operations
- Add the operation
- Add the request parameters
- Add the response values

Add Operation

Operations (optional)

GET
 PUT
 POST
 DELETE
 OPTIONS
 HEAD
 PATCH

REQUIRED	NAME	LOCATED IN	TYPE	DESCRIPTION
<input checked="" type="checkbox"/>	deposit	query ▾	float ▾	Annual deposits
<input checked="" type="checkbox"/>	rate	query ▾	float ▾	Interest rate
<input checked="" type="checkbox"/>	years	query ▾	int 64 ▾	Years of savings

Cancel
Add

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

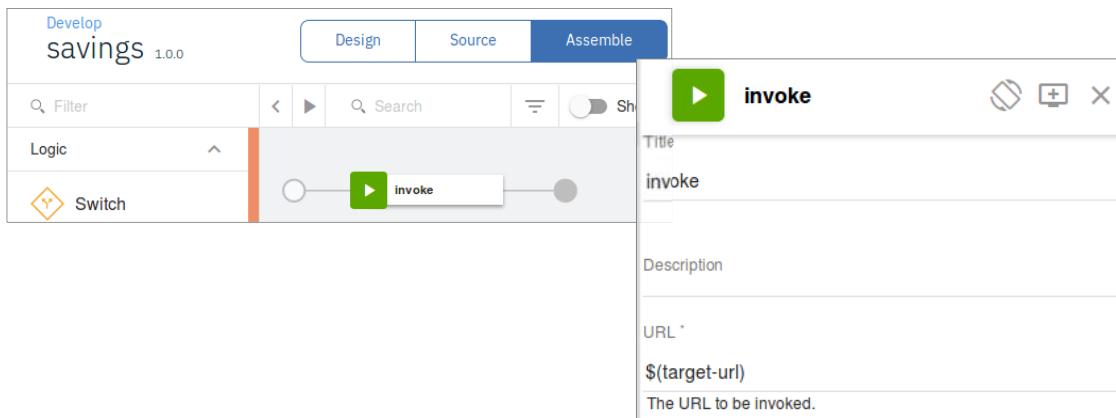
Figure 4-16. Step 6: Add a path and operations

You specified the location of the API implementation during the initial creation of the OpenAPI definition.

To complete the API definition, you must specify the exact interface for the API. The interface includes the API paths, HTTP operation, request, and response message for each API operation.

Step 7: Configure the gateway to call the REST application

- By default, the REST API definition forwards API requests to the API implementation
 - The `target-url` context variable is the network endpoint for the API application implementation
 - The `request.path` context variable represents the URL path in the original API request that starts with the base path of the API
 - The `api.operation.id` context variable represents the ID of the operation



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

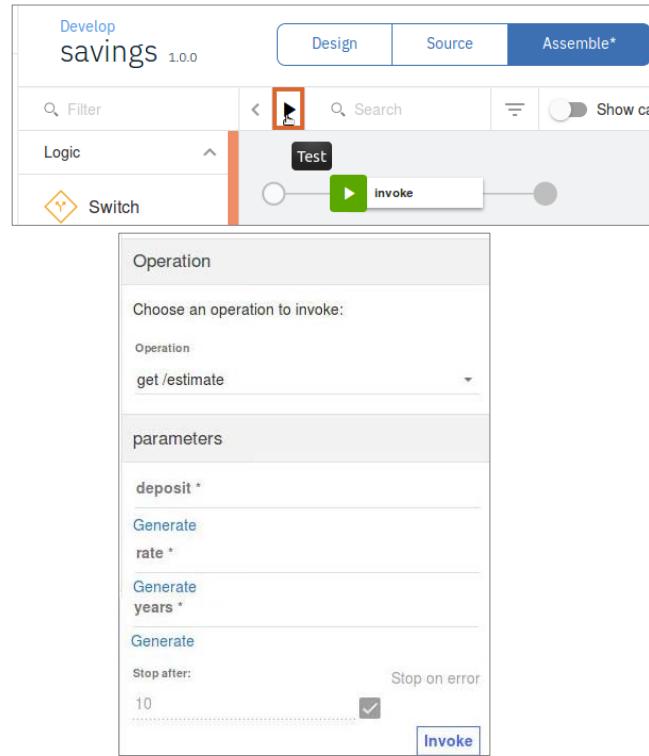
Figure 4-17. Step 7: Configure the gateway to call the REST application

After you define the API operations, you must instruct the API gateway to forward requests to the API application. The **policy assembly** is a set of message processing instructions that you define in an IBM extension section of an OpenAPI definition file. In the API Manager web application, open the **assemble** view to review the policies. Keep in mind that the policies apply to all operations that you defined in the API definition file.

By default, API Manager creates an assembly with one policy: the **invoke** policy. At run time, the API gateway makes an HTTP request to the endpoint in the **URL** field. This policy calls the remote endpoint with the API operation that the client sent to the gateway. The **target-url** context variable is the **target endpoint** value that you specify in API Manager. The **request path** context variable is the API path name for the operation. You can specify the actual endpoint address in the URL field of the assembly, or you can specify the URL as a combination of API Connect context variables.

Step 8: Test the API definition

- To test the REST API definition, use the test option of the assembly
 - The API Manager provides an option “Make API Online” that publishes the API definition to the gateway
 - You do not need to explicitly publish an API application for local testing in the assembly view, since the “Make API Online” feature does this function
- Review the API operations in the Test dialog
 - Use the test client to send a request to an API operation



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-18. Step 8: Test the API definition

To test the API endpoint that you defined in the OpenAPI definition, you must publish the API to a gateway.

To test the REST API click the test icon in the API Manager web application. The test dialog opens. The first time that you test an API, the test dialog prompts you to make the API online. By selecting this option, you publish the API and its product to the default gateway of the catalog. When the API is published, you can choose the operation that you want to call, provide the required parameters, and then click the **Invoke** button. The **Invoke** action calls the API operation that is hosted at the gateway.

4.2. Proxy an existing SOAP API

Proxy an existing SOAP API

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-19. Proxy an existing SOAP API

Topics

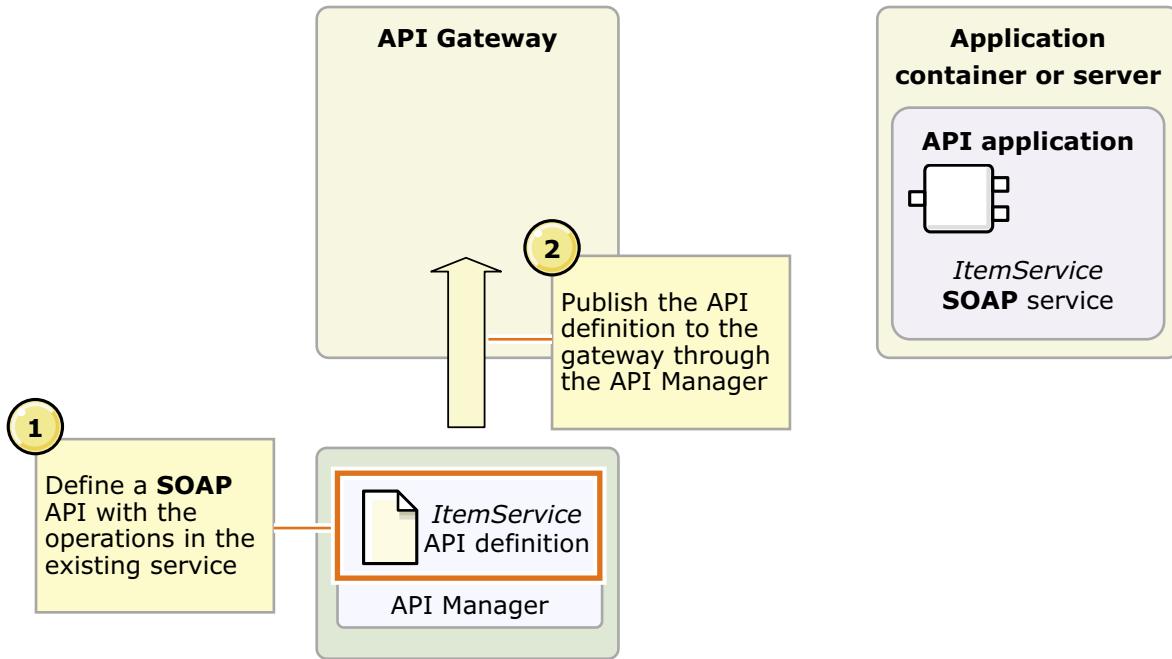
- Proxy an existing REST API
- ▶ Proxy an existing SOAP API

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-20. Topics

Scenario two: Define a SOAP API



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-21. Scenario two: Define a SOAP API

In the second scenario, your organization wants to expose an existing SOAP service at the API gateway. The goal is to forward API requests as-is to the SOAP service. In other words, you want to build a proxy for a SOAP service.

In the API Manager web application, build a new OpenAPI definition that is based on a SOAP interface. The most straightforward method is to select the open to create a SOAP proxy from an existing WSDL service in the API Manager web user interface. When you use this option, the dialog prompts you to import a Web Services Description Language (WSDL) document as a starting point for the API definition. After you import the WSDL interface and build the SOAP API definition, you auto-publish the API definition to a DataPower gateway in the API Manager. The auto-publish option happens when you make the API online in the API Manager test feature.

Scenario two: Gateway forwards SOAP API requests

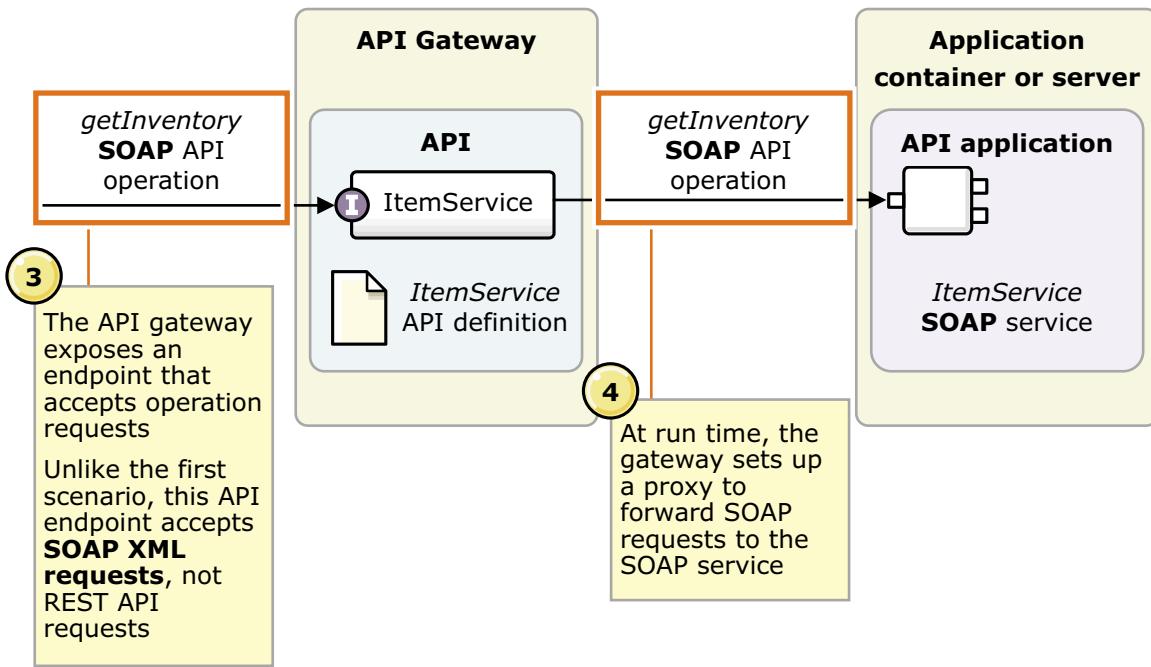


Figure 4-22. Scenario two: Gateway forwards SOAP API requests

To test the SOAP API that is hosted at the API gateway, send an SOAP request message to the gateway. You must use a test client that sends a properly formatted SOAP XML envelope message in the body of the HTTP request. The test client in the API Manager assembly view can build such a message for a SOAP API.

When the API gateway receives the SOAP request, it validates the message against the input message and parameters that you defined in the OpenAPI definition file. The **proxy** policy in the assembly forwards the request to the actual SOAP service implementation.

Step 1: Review the existing SOAP service

- Before you begin, examine the operations, messages, and XML schema types that the existing SOAP service describes
 - The WSDL document captures the SOAP service interface in a standard, structured form
 - Download a copy of the WSDL document
- In this example, test the existing service at:
<http://soapsample.mybluemix.net>

ItemService SOAP API Sample

This test client invokes the `getInventory` operation from the SOAP service.

SOAP request:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><soapenv:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:tns="http://soapsample.training.ibm.com/"><soapenv:Body>
<tns:getInventory></tns:getInventory></soapenv:Body></soapenv:Envelope>
```

SOAP response:

HTTP status code 200

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body><ns2:geInventoryResponse
  xmlns:ns2="http://soapsample.training.ibm.com/"><return><ns2:name>Dayton
  Meat Chopper</ns2:name><ns2:description>Punched-card tabulating machines and
  time clocks were not the only products offered by the young IBM. Seen here
  in 1930, manufacturing employees of IBM's Dayton Scale Company are
  assembling Dayton Safety Electric Meat Choppers.</ns2:description>
<ns2:img>images/items/meat-chopper.jpg</ns2:img><ns2:img_alt>Meat
  Chopper</ns2:img_alt><ns2:id>1</ns2:id></return><return><ns2:name>Hollerith
  Tabulator and Sorter</ns2:name><ns2:description>This equipment is
  representative of the tabulating system invented and built for the U.S.
  Census Bureau by Herman Hollerith (1860-1929). After observing a train
  conductor punching railroad tickets to identify passengers, Hollerith
  conceived and developed the idea of using punched holes to record facts
  about people. These machines were first used in compiling the 1890 Census.
</ns2:description><ns2:img>images/items/hollerith-tabulator.jpg</ns2:img>
<ns2:img_alt>Tabulator</ns2:img_alt><ns2:id>2</ns2:id></return><return>
<ns2:name>IBM 77 Electric Punched Card Collator</ns2:name>
```

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-23. Step 1: Review the existing SOAP service

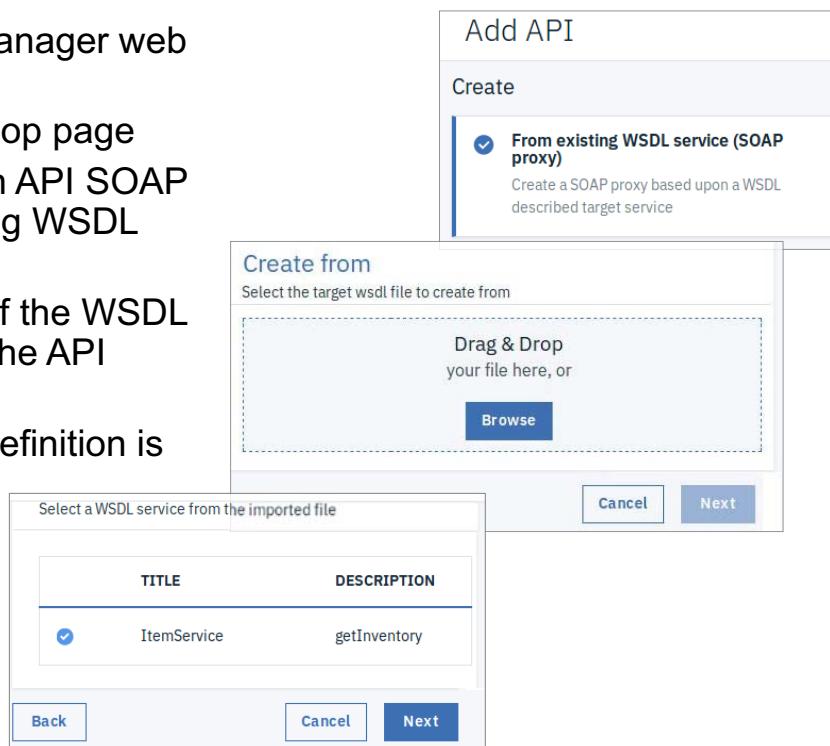
In this example, you create an OpenAPI definition for an existing SOAP service. The **ItemService** SOAP API sample returns a set of inventory items in the `getInventory` SOAP operation. You can try out the ItemService SOAP API from the <http://soapsample.mybluemix.net> website.

Notice that SOAP services have a specific way to format input parameters and the operation response. The SOAP specification defines an XML message format that is known as the SOAP envelope that encapsulates parameters, results, and fault messages. The OpenAPI definition must define how to create the SOAP request and response messages for a client application.

For more information about the SOAP specification, see: <https://www.w3.org/TR/soap/>

Step 2: Create a SOAP API definition in API Manager

1. Start the API Manager web application
2. Open the Develop page
3. Create an Open API SOAP proxy an existing WSDL SOAP service
4. Import a copy of the WSDL document into the API definition
5. The OpenAPI definition is created



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-24. Step 2: Create a SOAP API definition in API Manager

Start the API Manager web application and click the Develop APIs and Products tile. Select the **Add** option on the page. Then, select API. On the Add API page, select the option to create an OpenAPI document based on an existing WSDL service.

Upload the WSDL document from a local directory. In this example, you downloaded a copy of the ItemService WSDL document from the <http://soapsample.mybluemix.net/ItemService?WSDL> link. The ItemService is uploaded into the API definition. The web application generates the proper OpenAPI paths, operations, and schema types based on the WSDL document.

Step 3: Review the API operations

- The API Manager application creates a set of **API operations** based on the **portType** section in the WSDL document
 - For each WSDL operation, the API definition declares a **POST** API operation
 - The request and response message map to the **WSDL input** and **output** messages

The screenshot shows the 'Operations' section of the API Manager interface. A new operation named '/getInventory' has been created with the HTTP method 'POST'. Below this, the 'Parameters' section shows a single parameter named 'body' located in 'body' with type 'getInventoryInput'. In the 'Response' section, there is a default response with status code 'default' and schema 'getInventoryOutput'.

REQUIRED	NAME	LOCATED IN	TYPE	DESCRIPTION
<input checked="" type="checkbox"/>	body	body	getInventoryInput	

STATUS CODE	SCHEMA	DESCRIPTION
default	getInventoryOutput	

Figure 4-25. Step 3: Review the API operations

Review the API paths and operations. For each operation in the WSDL port type, the create API dialog creates a matching API operation. The API path corresponds to the name of the operation. The HTTP method is always set to **POST** because SOAP services do not use the other HTTP methods, such as GET, PUT, or DELETE.

In each API operation, the request and response messages map to the WSDL input and output message types.

Step 4: Review the request and response type definitions

Definitions		
API request and response payload structures are composed using OpenAPI schema definitions.		
NAME	TYPE	DESCRIPTION
Security	object	Header for WS-Security
getInventoryInput	object	⋮
getInventoryHeader	object	⋮
getInventoryOutput	object	⋮
getInventory_tns	object	⋮
getInventoryResponse_tns	object	⋮
item_tns_unqual	object	⋮

```

getInventoryInput:
  type: object
  properties:
    Envelope:
      xml:
        namespace: 'http://schemas.xmlsoap.org/soap/envelope/'
        prefix: soapenv
    type: object
    properties:
      Header:
        $ref: '#/definitions/getInventoryHeader'
      Body:
        xml:
          namespace: 'http://schemas.xmlsoap.org/soap/envelope/'
          prefix: soapenv
          type: object
          properties:
            getInventory:
              $ref: '#/definitions/getInventory_tns'
            required:
              - getInventory
          required:
            - Body
        required:
          - Envelope
      example: >-
  
```

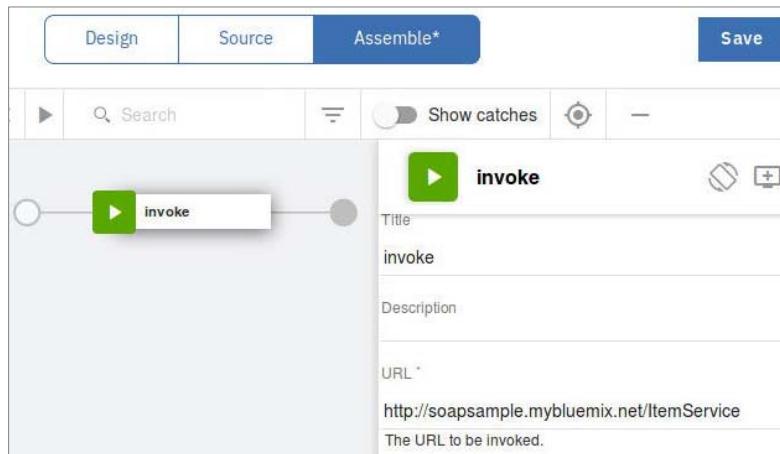
Figure 4-26. Step 4: Review the request and response type definitions

In REST APIs, the API path represents the name of the operation. However, SOAP APIs encode the operation name into the XML SOAP envelope in the HTTP request message. The OpenAPI definition must capture the same information in the schema definition section.

In this example, the **getInventory** OpenAPI operation represents the WSDL operation of the same name. In the original WSDL document, the `getInventory` operation accepts a SOAP request message named “`getInventory`”. The OpenAPI definition creates a custom schema type named “`getInventory`” to represent this data. The **getInventoryInput** schema type represents the SOAP envelope for the request message. The **getInventoryOutput** schema type represents the SOAP envelope for the response message.

The WSDL operation, input, output, and custom data types correspond to the OpenAPI schema definitions.

Step 5: Review the invoke policy in the assembly flow



- The **invoke** policy forwards the original SOAP API request to the actual SOAP service implementation
- The API Manager sets the **target URL** setting in the invoke policy to the WSDL service endpoint address
 - For example, <http://soapsample.mybluemix.net/ItemService> is the address for the ItemService SOAP API implementation

Figure 4-27. Step 5: Review the invoke policy in the assembly flow

For a SOAP proxy policy, the **invoke** policy forwards the message payload in the assembly flow to the **URL** endpoint location. The API gateway can run one invoke policy per flow – you cannot call multiple invoke policies for SOAP proxies in the same flow.

Step 6: Test the SOAP API in the API Manager

- Test the SOAP API from the test feature in the assembly view
- Choose the operation to invoke
- Supply required SOAP XML test data in the body area of the request message
- Click **Invoke** to call the SOAP proxy

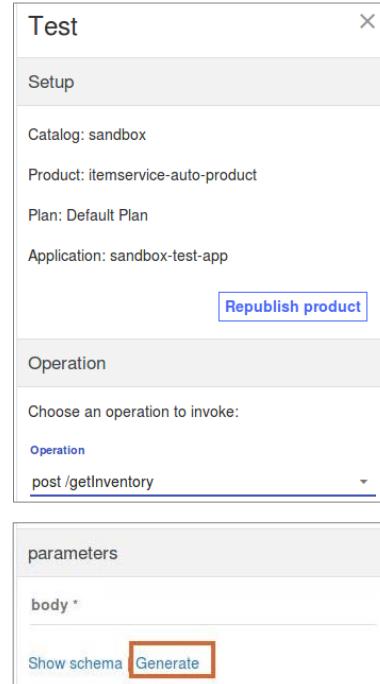


Figure 4-28. Step 6: Test the SOAP API in the API Manager

Test the SOAP API by clicking the Test icon in the assembly editor in API Manager.

The first time the test is run, click the “Make API Online” option in the test dialog. This option publishes the API to the gateway. When the “Activate API” option is selected when the API is first created, the API is already published to the gateway. In this case, the “Republish product” option is displayed in the test dialog.

Select the operation and generate some sample data in the SOAP body area of the test feature. Finally, click **Invoke** to call the SOAP API.

Step 7: Response message from the SOAP API call

- The response message from the successful call of the SOAP proxy is displayed

Response	
Status code:	200 OK
Response time:	635ms
Headers:	<pre>apim-debug-trans-id: 426530149-Landlord- apiconnect-23cb75e3-b599-45de- bd45-65556c196c06 content-language: en-US content-type: text/xml; charset=UTF-8 x-global-transaction- id: 196c55655c8ac78b00000a32 x-ratelimit-limit: name=rate-limit,100; x-ratelimit-remaining: name=rate-limit,99;</pre>
Body:	<pre><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> <soap:Body> <ns2:geInventoryResponse xmlns:ns2="http://soapsample.training.ibm.com/"> <return> <ns2:name>Dayton Meat Chopper</ns2:name> <ns2:description>Punch ed-card tabulating machines and time c locks were not the only products offered by the young IBM. Seen here in 1930 , manufacturing employees of IBM's Dayton Scale Company are assembling Dayton Safety Electric Meat Choppers.</ns2:description> <ns2:img>images/items/meat-chopper.jpg</ns2:img> <ns2:img_alt>Meat Chopper</ns2:img_alt> <ns2:id>1</ns2:id> </return></pre>

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-29. Step 7: Response message from the SOAP API call

The response message returns a successful call of the DataPower SOAP proxy policy.

The example shows the response message from a successful call of the SOAP proxy on the DataPower gateway.

Unit summary

- Explain the role of the API gateway in exposing existing services
- Explain how to expose an existing REST service in an API definition
- Explain how to expose an existing SOAP service in an API definition

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-30. Unit summary

Review questions

1. True or False: Use the API Manager user interface to implement APIs.
2. How do you define an SOAP API Definition in API Connect?
 - A. Import a WSDL document in API Manager
 - B. Create a SOAP API from the assembly palette
 - C. Generate a sample with the apic utility
 - D. Import a SOAP application in API Designer
3. Which environment variable represents the endpoint of the API application?
 - A. \$(host)
 - B. \$(api.port)
 - C. \$(target-url)
 - D. \$(host.url)



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-31. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: Use the API Manager user interface to implement APIs.
The answer is False. Use the API Connect developer toolkit with LoopBack to implement APIs.
2. How do you define an SOAP API Definition in API Connect?
 - A. [Import a WSDL document in API Manager](#)
 - B. Create a SOAP API from the assembly palette
 - C. Generate a sample with the apic utility
 - D. Import a SOAP application in API Designer**The answer is A.**
3. Which environment variable represents the endpoint of the API application?
 - A. `$ (host)`
 - B. `$ (api.port)`
 - C. [`\$ \(target-url\)`](#)
 - D. `$ (host.url)`**The answer is C.**

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-32. Review answers



Exercise: Define an API that calls an existing SOAP service

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-33. Exercise: Define an API that calls an existing SOAP service

Exercise objectives

- Review the SOAP sample
- Download the WSDL file
- Create an API definition that invokes an existing WSDL service
- Review the assembly in API Manager
- Test the SOAP API on the DataPower gateway



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2020

Figure 4-34. Exercise objectives

Unit 5. Implementing APIs with the LoopBack framework

Estimated time

01:00

Overview

This unit introduces the LoopBack framework, which is a model-driven approach to building REST APIs in NodeJS. You examine the different types of LoopBack connectors and how they support the generation of application code. Learn how to generate an application scaffold with the Developer Toolkit. Implement an API that calls a REST back-end service with the LoopBack REST connector.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Describe what is LoopBack
- Describe the function of LoopBack connectors and code generation
- Create the application scaffold from the apic command-line utility
- Implement an API with a NodeJS LoopBack application
- Test the LoopBack application and operation with the LoopBack Explorer

5.1. Introduction to the LoopBack framework

Introduction to the LoopBack framework

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-2. Introduction to the LoopBack framework

Topics

-  Introduction to the LoopBack framework
 - Implement an API with a LoopBack application

Example: Product inventory and price calculator

- **Example 1:** Customer accounts
 - You have an internal web service that lists details on customer accounts
 - You want to allow application developers outside your company to retrieve the customer account on behalf of a customer
- Define an API in the gateway that proxies service requests to the existing API in **API Manager**
- **Example 2:** Product pricing
 - You want to update the price of products based on inventory levels and customer demand
 - The cost of the product is saved in an existing database
- To display the product inventory, create an API that calls an existing API implementation with the **Loopback REST connector**

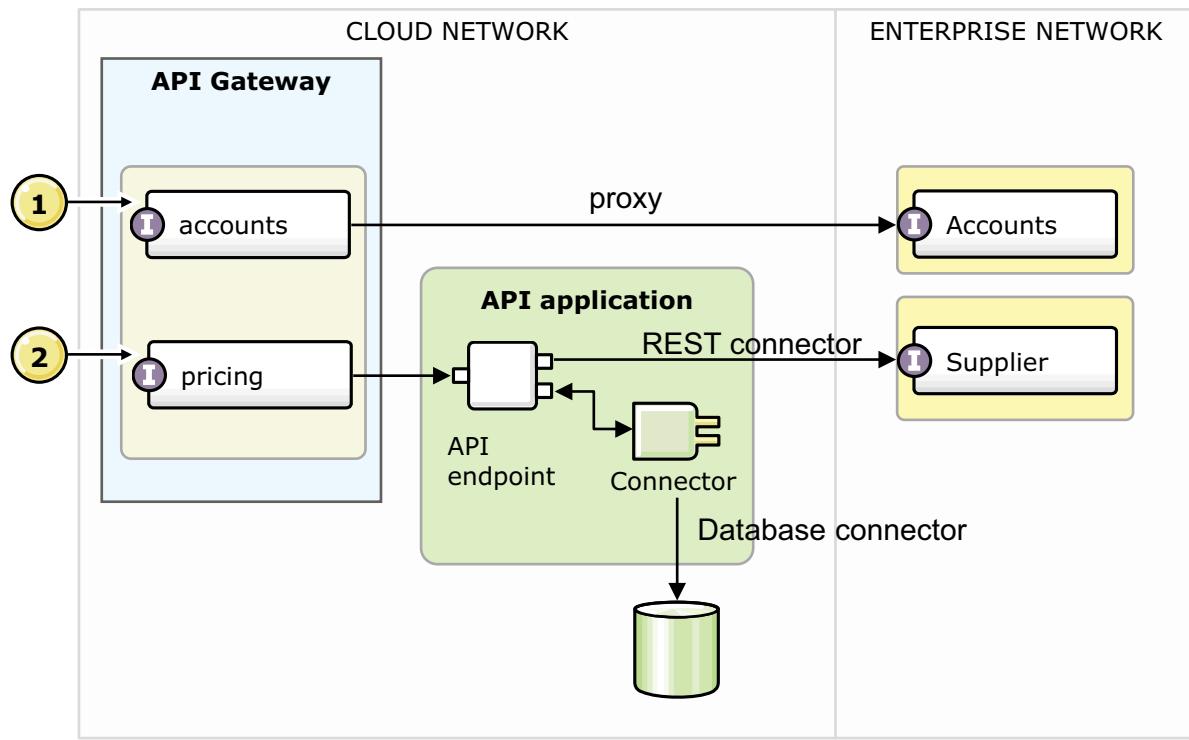
Figure 5-4. Example: Product inventory and price calculator

In the first example, your company hosts a customer account API in your system of record. You want to extend access for the existing API to Business Partner and third-party applications outside of your company. You determined that the customer API can be used as is.

In the second example, your company hosts an API that gets a list of products based on inventory levels in a third-party back-end application. To abstract these details from your API consumers, you build an API that calls an existing API implementation.

You create a new model and API in LoopBack from the existing API implementation by using the REST connector. The new product listing exposes only the GET (find) method from the existing back-end implementation.

Message flows in IBM API Connect



Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

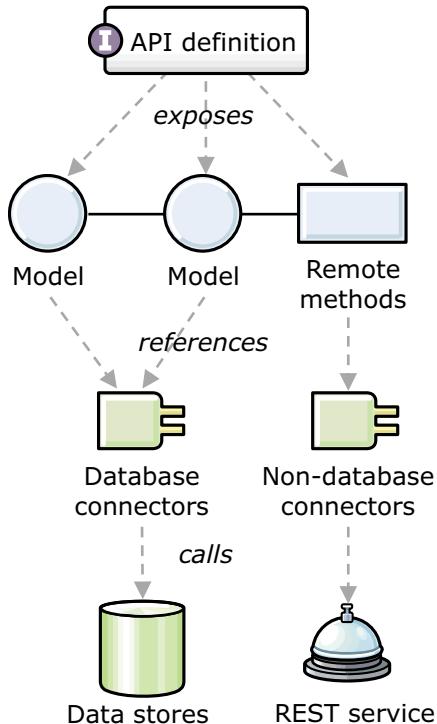
Figure 5-5. Message flows in IBM API Connect

This diagram illustrates how APIs and existing APIs fit in a solution with IBM API Connect.

1. When an API consumer calls the accounts API, the API gateway forwards the request to an existing API. The Accounts service in the system of record returns the data back to the gateway. In this scenario, the Gateway passes the data to the consumer unchanged.
2. When an API consumer calls the pricing API, the API gateway calls an API. The API calls one or more existing APIs and data sources. It manipulates the data from these sources with its own logic. The purpose of an API is to reuse existing sources of data and services with new business logic.

What is LoopBack?

- LoopBack is a Node.js framework for building APIs
- You define business models, properties, and relationships in your LoopBack application
 - The LoopBack framework creates a set of REST API operations that give client access to these model objects
- You configure a data source from a LoopBack connector to access and persist model data
- You can also develop remote methods: API operations that do not map a data access method to a model



[Implementing APIs with the LoopBack framework](#)

© Copyright IBM Corporation 2020

Figure 5-6. What is LoopBack?

LoopBack is an open source application framework for implementing APIs in Node.js. In IBM API Connect, you develop LoopBack applications to implement APIs: stand-alone APIs in the Cloud network layer. In the following slide, you examine how APIs integrate with existing services at the system API layer.

With LoopBack, you define business models, properties, and relationships through command-line tools. The LoopBack framework creates a set of predefined operations to create, retrieve, update, and delete models. The framework also persists the model data to a data source through a database connector.

Your LoopBack application can also call other remote services and APIs through non-database connectors.

You can also implement your own free-form API operation with a remote method.

With the LoopBack framework, you can quickly implement your API with generation tools, configuration files, and a minimal set of code.

LoopBack data source connectors

- Built-in memory connector
 - Acts like a database connector, in that it supports standard query and create, read, update, and delete operations
 - Used for local development and testing
- Database connectors
 - Connect to relational and NoSQL databases
- Other connectors
 - Connect to REST or SOAP APIs and other backend systems

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-7. LoopBack data source connectors

LoopBack connectors implement the data exchange logic to communicate with backend systems such as relational or NoSQL databases, or other connectors, such as the REST or SOAP API.

Officially supported LoopBack connectors include:

Built-in memory connector

Acts like a database connector, in that it supports standard query and create, read, update, and delete operations

Used for local development and testing

Database connectors

Connect to relational and NoSQL databases

Other connectors

Connect to REST or SOAP APIs and other backend systems

LoopBack database connectors

- LoopBack database connectors implement create, retrieve, update, and delete operations as a common set of methods of PersistedModel
- When you attach a model to a data source backed by one of the database connectors, the **model automatically acquires the create, retrieve, update, and delete methods** from PersistedModel
- The data access methods on a persisted model are exposed to REST by default
- Some of the officially supported database connectors include:
 - Cassandra connector
 - Cloudant connector
 - DashDB
 - DB2 connector
 - DB2 for iSeries connector
 - DB2 for z/OS
 - Informix connector
 - **MongoDB connector**
 - **MySQL connector**

The case study in the exercises that accompany this course use the MongoDB connector and the MySQL connector to provide examples of connecting to non-relational and relational data sources

Figure 5-8. LoopBack database connectors

LoopBack database connectors implement create, retrieve, update, and delete operations as a common set of methods that are inherited from the PersistedModel. When you attach a model to a data source backed by one of the database connectors, the model automatically acquires the create, retrieve, update, and delete methods from PersistedModel. The data access methods on a persisted model are exposed to REST by default.

The case study in the exercises that accompany this course use the MongoDB connector and the MySQL connector to provide examples of connecting to NoSQL and SQL (relational) data sources.

LoopBack connectors to back-end systems

- Models attach to **non-database sources**
 - Models do not have properties
 - Implement specific methods that depend on the underlying system
 - Read-only
- Available connectors:
 - Email connector
 - JSON RPC connector
 - MQ Light connector
 - Push connector
 - Remote connector
 - REST connector
 - SOAP connector
 - Storage connector
 - Swagger connector

The example LoopBack application that follows in the presentation unit uses the REST connector to interact with other third-party REST APIs

Figure 5-9. LoopBack connectors to back-end systems

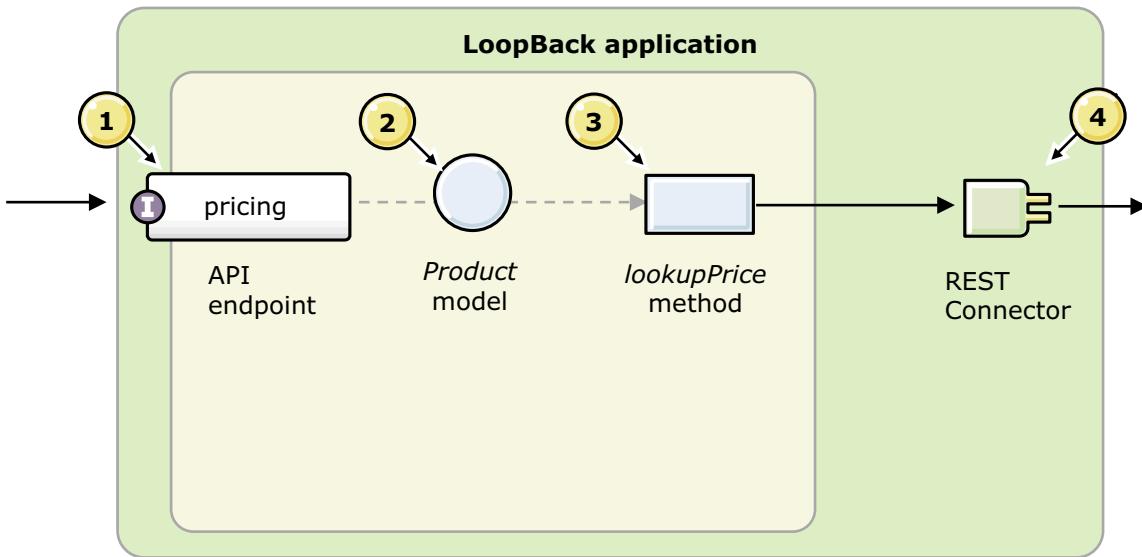
LoopBack supports a number of connectors to backend systems beyond databases.

These types of connectors often implement specific methods that depend on the underlying system. For example, the REST connector delegates calls to REST APIs.

Models attached to non-database data sources can serve as controllers (a model class that has only methods). Such models usually do not have property definitions as the backing connector does not support create, read, update, and delete operations.

The list of supported non-database connectors for LoopBack V3 is shown on the page.

Implement an API with a NodeJS LoopBack application



Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-10. Implement an API with a NodeJS LoopBack application

This diagram explains the main components in a LoopBack application. You examine each of these components in greater detail in later units and exercises.

1. The API definition OpenAPI file specifies the API operations that the LoopBack application makes available.
2. You define the business data and logic in the model objects, or **models**. The LoopBack framework automatically creates a set of create, retrieve, update, and delete operations for each Persisted model in your application that uses a database connector. The dashed lines denote the link between API operations and the model objects.
3. Non-database connectors do not automatically create a set of create, retrieve, update, and delete API operations. To create your own operation, define a **remote method** within a model object.
4. Your LoopBack application can call system APIs, services from outside your company, or other APIs. LoopBack provides a set of **non-database connectors** to make it easier for you to call these services.

5.2. Implement an API with a LoopBack application

Implement an API with a LoopBack application

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-11. Implement an API with a LoopBack application

Topics

- Introduction to the LoopBack framework
- ▶ Implement an API with a LoopBack application

Create a LoopBack application: example

In this scenario, you implement an API with the LoopBack framework

1. Generate a LoopBack application with the API Connect toolkit
2. Define a REST connector data source
3. Configure the data source that supplies information to your application
4. Create the models, properties, and relationships in the LoopBack application
5. Implement the API logic in the model object
6. Start the Loopback application and the LoopBack Explorer
7. Review the application and operations in the LoopBack web application
8. Test the operation in the Explorer
9. Review the responses from calling the API with the LoopBack REST connector



[Implementing APIs with the LoopBack framework](#)

© Copyright IBM Corporation 2020

Figure 5-13. Create a LoopBack application: example

In this topic, you review an example on how to implement an API with the IBM API Connect Toolkit. Students are encouraged to create this LoopBack example on the remote lab platform.



Information

Before you start, it is suggested that you review the REST API back-end application that the LoopBack application connects to with the REST connector.

The back-end application is an IBM Cloud example with the address
<https://pricesample.mybluemix.net>

You can call the IBM Cloud example application with the URL:

<https://pricesample.mybluemix.net/price/{id}>, where {id} is a value in the range 1 – 6.

Sample response in JSON format:

```
{"product-id": "1", "price": 100.01}
```

Step 1: Generate a LoopBack application

The IBM API Connect toolkit is installed on the student lab environment. Use the command-line LoopBack generation tool to create the application scaffold

1. Open a terminal
2. Run the LoopBack application generator

```
$ apic lb
? What's the name of your application? pricing
? Enter name of the directory to contain the project: pricing
    Info change the working directory to inventory
? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind?
    empty-server API, without any configured models or data
sources
```

3. Change directory into your LoopBack application

```
$ cd pricing
```

Figure 5-14. Step 1: Generate a LoopBack application

Step 1: Generate a LoopBack application.

After you install the IBM API Connect toolkit, use the command line LoopBack generation tool to create the application scaffold.

1. Open a terminal (Linux).
2. Run the LoopBack application generator.
3. Change directory into your LoopBack application.

The `apic lb` command creates the directory structure and a set of configuration files in a directory. By default, the directory is the same name as your application name.

You must complete the rest of the steps in the LoopBack directory. If you call the `apic` commands in another directory, they do not work.

Step 2: Define a data source

Before you define your model objects, configure the data sources that supply data to your models. Ensure that you are in the `pricing` directory

1. Create a LoopBack data source

```
$ apic lb datasource
? Enter the data-source name: priceREST
? Select the connector for priceREST:
  REST services (supported by StrongLoop)
Connector-specific configuration:
? Base URL for the REST service:
  https://pricesample.mybluemix.net/price/
? Default options for the request:
? An array of operation templates:
? Use default CRUD mapping: No
? Install loopback-connector-rest@^3.2.0 Yes
```

2. Review and edit the data source settings

```
$ gedit server/datasources.json
```

Figure 5-15. Step 2: Define a data source

Step 2: Define a data source.

Before you define your model objects, configure the data sources that supply data to your models.

1. Create a LoopBack data source.
2. Review and edit the data source settings.

If you create the LoopBack data source after you create your model, you must edit the `datasources.json` configuration file and bind a data source to each model object.

Step 3: Configure the data source

- In this example, you map the existing REST service to the LoopBack REST data source in `~/pricing/server/datasources.json`
- Overwrite the generated `datasources.json` with the syntax shown:

```
{
  "priceREST": {
    "name": "priceREST",
    "crud": false,
    "connector": "rest",
    "operations": [
      {
        "template": {
          "method": "GET",
          "url": "https://pricesample.mybluemix.net/price/{id}",
          "headers": {
            "accepts": "application/json",
            "content-type": "application/json"
          }
        },
        "functions": {
          "price": ["id"]
        }
      ]
    }
  }
}
```

server/datasources.json

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-16. Step 3: Configure the data source

Step 3: Manually configure the REST connector data source.

Depending on which data source you use, you might have to add more configuration information in the `server/datasources.json` file.

In this example, you defined a REST data source that is named “`priceREST`”. After you generated the data source from the `apic` command-line utility, you must specify two fields: **template** and **functions**. The **template** object specifies the external REST service that you want to call. You specify the network endpoint in the **URL** field, the HTTP **method** to call, and the HTTP request **headers**.

The **functions** object specifies the LoopBack function.

The **price** function takes an input parameter named **id**. This function makes an REST API call to `https://pricesample.mybluemix.net/price/{id}`, substituting the input parameter for the `{id}` path.

For more information about the REST connector configuration settings, see:
<http://loopback.io/doc/en/lb3/REST-connector.html>

Code snippets can be copied from the `/home/localuser/lab_files/demos/pricing` folder on the student image.



Syntax

```
{  
  "priceREST": {  
    "name": "priceREST",  
    "crud": false,  
    "connector": "rest",  
    "operations": [  
      {  
        "template": {  
          "method": "GET",  
          "url": "https://pricesample.mybluemix.net/price/{id}",  
          "headers": {  
            "accepts": "application/json",  
            "content-type": "application/json"  
          }  
        },  
        "functions": {  
          "price": ["id"]  
        }  
      }  
    ]  
  }  
}
```

Step 4: Create models, properties, and relationships

- In a LoopBack application, **model** objects represent business data
 - The LoopBack framework retrieves and persists data with **data sources**
 - The LoopBack framework creates a set of API operations that API consumers use to access model data

1. Create a LoopBack model

```
$ apic lb model
? Enter the model name: product
? Select the data-source to attach undefined to:
  priceREST (rest)
? Select model's base class Model
? Expose product via the REST API? Yes
? Custom plural form (used to build REST URL):
? Common model or server only? Common

Let's add some product properties now.
Enter an empty property name when done.
? Property name:

Done running loopback generator
```

Figure 5-17. Step 4: Create models, properties, and relationships

Step 4: Create models, properties, and relationships.

- In a LoopBack application, model objects represent business data.
- The LoopBack framework retrieves and persists data with data sources.
- The LoopBack framework creates a set of API operations that API consumers use to access model data.

Run the `apic lb model` command to generate a LoopBack model in the terminal.

Step 5: Implement API logic with a remote method

- Implement the message processing logic in the API in Node.js
 - Overwrite the code in
~/pricing/common/models/product.js

```
'use strict';
module.exports = function(Product) {
  var priceService;
  Product.on('attached', function() {
    priceService = Product.app.dataSources.priceREST;
  });
  Product.lookupPrice = function() {
    priceService.price.apply(priceService, arguments);
  };
  Product.remoteMethod('lookupPrice', {
    description: 'Calculate price for specified product',
    accepts: [{ arg: 'productId', type: 'number' }],
    returns: { arg: 'price', root: true },
    http: { verb: 'get' }
  );
};
```

common/models/product.js****

Figure 5-18. Step 5: Implement API logic with a remote method

Step 5: Implement API logic.

For each model object that you define, the apic command line utility generates two files: the model definition file, and the model script file. The default location for model script and definition files is the common/models directory.

The model definition file stores the name, properties, and relationships of the LoopBack model in a JSON format. In this example, the name of the model definition file is `product.json`.

The model script file stores custom methods that you define for the model. The name of the model script file in this example is `product.js`.

In this example, you defined a remote method that is named `lookupPrice` in the `Product` model object. The method takes a product identifier as an input parameter, and returns the price of the product. You implemented the logic for the API operation in the `Product.lookupPrice` JavaScript function.

Code snippets can be copied from the `/home/localuser/lab_files/demos/pricing` folder on the student image.



Syntax

```
module.exports = function(Product) {
  var priceService;
  Product.on('attached', function() {
    priceService = Product.app.dataSources.priceREST;
  });
  Product.lookupPrice = function() {
    priceService.price.apply(priceService, arguments);
  };
  Product.remoteMethod('lookupPrice', {
    description: 'Calculate price for specified product',
    accepts: [{ arg: 'productId', type: 'number' }],
    returns: { arg: 'price', root: true },
    http: { verb: 'get' }
  });
};
```

Step 6: Start the LoopBack application and Explorer

- To test your API, you start these components:
 - The **LoopBack application** implements the API
 - The **LoopBack Explorer** browser-based test function
- Start the LoopBack application from the pricing directory:
 - From the terminal type:

```
$ cd ~/pricing
$ npm start
> pricing@1.0.0 start/home/localuser/pricing
> Node .
Web server listening at http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- The LoopBack NodeJS application and the LoopBack Explorer start
- Copy the URL with the LoopBack Explorer value into the address in a browser session

Figure 5-19. Step 6: Start the LoopBack application and Explorer

Step 6: Start the LoopBack application and the Loopback Explorer.

Start the LoopBack application by typing “npm start” in the folder where the LoopBack application was generated.

The LoopBack application starts and the LoopBack Explorer also starts automatically.

Copy the URL for the Explorer into the address area of your browser.

Step 7: Review the application in the Loopback Explorer

- Use the LoopBack Explorer as a test client for your interaction APIs
- Click **Show/Hide** or **List Operations** in the Explorer to view the operations
 - The list of operations for the pricing application is displayed on the page in the browser

The screenshot shows the LoopBack API Explorer interface. At the top, there's a header bar with the title "LoopBack API Explorer" and a URL field showing "localhost:3000/explorer/#/". Below the header, a green navigation bar has the title "LoopBack API Explorer" and a "Token Not Set" message. There are buttons for "accessToken" and "Set Access Token". The main content area is titled "pricing" and contains a single entry: "pricing". Under "pricing", the "product" section is expanded, showing three operations:

Method	Path	Description
POST	/products/invoke	
GET	/products/lookupPrice	Calculate price for specified product
GET	/products/price/{id}	

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-20. Step 7: Review the application in the Loopback Explorer

Step 7: Review the application in the **LoopBack Explorer**.

The LoopBack Explorer is the integral test client for LoopBack applications.

Run the LoopBack Explorer by typing the address in the browser. The URL for the Explorer is displayed in the terminal that runs the application.

Click either the Show/Hide or the List Operations in the Explorer to display the operations for the application.

Select the operation that you want to test. In this example, the operation is named GET /products/lookupPrice.

Step 8: Test the operation in the Loopback Explorer

- When you select the operation, the page displays example responses and the required request parameters
- Type a value for the `productId` request parameter.
Then, click **Try it out**

The screenshot shows the Loopback Explorer interface for the `/products/lookupPrice` operation. At the top, there's a header with "GET" and the endpoint `/products/lookupPrice`. To the right of the endpoint is a descriptive text: "Calculate price for specified product". Below the header, there's a section titled "Response Class (Status 200)" with the message "Request was successful". There are two tabs: "Model" and "Example Value", with "Example Value" being active. The "Example Value" field contains an empty JSON object: `{}
`. Underneath, there's a "Response Content Type" dropdown set to "application/json". Below that is a "Parameters" table:

Parameter	Value	Description	Parameter Type	Data Type
<code>productId</code>	1		query	double

At the bottom left of the interface is a prominent "Try it out!" button.

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-21. Step 8: Test the operation in the Loopback Explorer

Step 8: Test the API operation with the **LoopBack Explorer**.

When you select the operation, the page displays example responses and the required request parameters

Type a value in the range of 1 – 6 for the `productId` request parameter.
Then, click **Try it out**.

Step 9: Review the results

- The response is displayed in the Explorer
- A successful response is returned from the back-end with the REST connector

```
Curl
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/products/lookupPrice?productId=1'

Request URL
http://localhost:3000/api/products/lookupPrice?productId=1

Response Body
{
  "product-id": "1",
  "price": 100.01
}

Response Code
200
```

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-22. Step 9: Review the results

Step 9: Review the response message in the Explorer.

A response code 200 and the response body are returned in the Explorer.

You successfully called the back-end API from the LoopBack application with the REST connector.



Information

The IBM Cloud sample at

<https://lookup-price.mybluemix.net/api/products/lookupPrice?productId={id}> does the same REST call that you created in this unit.

Relationship between API gateway and LoopBack

- The LoopBack application implements the *functional requirements* of your API
 - LoopBack creates a set of data-centric create, retrieve, update, and delete API operations for each model that you define
 - You can also implement custom free-form APIs as remote methods
- The API gateway enforces the *non-functional requirements* of your API
 - You define a set of security, control, and message-processing policies for your API with the API Designer web application
 - The API gateway applies rate limit, access control, message mapping, and other policies before it calls the LoopBack application

Figure 5-23. Relationship between API gateway and LoopBack

The LoopBack application is the implementation for your API.

When you call an API operation, the LoopBack framework finds the model method that corresponds to the API operation. Most of the methods are built in to LoopBack, for example, when you define a LoopBack model that is named *item* with two properties: *name* and *price*. When you call `GET /api/item`, you retrieve a JSON object with a name and price. No coding is required to build these APIs if you use one of the Loopback database connectors.

In short, the built-in create, retrieve, update, and delete operations and custom remote methods represent the business logic in your API. The LoopBack application implements the functional requirements of your application.

Unit summary

- Describe what is LoopBack
- Describe the function of LoopBack connectors and code generation
- Create the application scaffold from the apic command line utility
- Implement an API with a NodeJS LoopBack application
- Test the LoopBack application and operation with the LoopBack Explorer

Review questions

1. True or False: You deploy your LoopBack application to the API Gateway.
2. True or False: You must create a LoopBack application to edit the API interface definition in the API Manager web application.
3. True or False: In IBM API Connect, a LoopBack application is a way to implement an API.



Figure 5-25. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: You deploy your LoopBack application to the API Gateway. **The answer is False. Although you can publish an endpoint for the API on the gateway, you deploy the LoopBack API application as a separate NodeJS application that runs locally or in a container.**
The gateway does not host the LoopBack application.

2. True or False: You must create a LoopBack application to edit the API interface definition in the API Manager web application.
The answer is False. You can create a OpenAPI definition in API Manager without creating a LoopBack application.

3. True or False: In IBM API Connect, a LoopBack application is a way to implement an API.
The answer is True. A LoopBack application is an option for implementing APIs.



Exercise: Create a LoopBack application

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2020

Figure 5-27. Exercise: Create a LoopBack application

Exercise objectives

- Create an application scaffold with the apic command-line utility
- Examine LoopBack models and properties
- Test the application with the Loopback API Explorer browser-based test client



Figure 5-28. Exercise objectives

Unit 6. LoopBack models, properties, and relationships

Estimated time

01:15

Overview

This unit explores the concepts of LoopBack models, properties, and relationships. You learn how to define models and properties to map business data structures to API resources. You also learn how to create relationships between models.

How you will check your progress

- Review questions

Unit objectives

- Explain the relationship between the model and the API
- Explain the concepts of LoopBack models, properties, and relationships
- Explain the features that each model base class inherits
- Identify the property data types
- List the model relationship types
- Define models, properties, and relationships in the API Connect Developer toolkit apic utility

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-1. Unit objectives

6.1. LoopBack framework in API Connect

LoopBack framework in API Connect

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-2. LoopBack framework in API Connect

Topics

- ▶ LoopBack framework in API Connect
 - LoopBack models, properties, and relationships

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-3. Topics

LoopBack: An open source Node.js API framework

- LoopBack is an open source Node.js application framework for REST APIs
 - For more information, see: <http://loopback.io/doc>
- The LoopBack framework makes a set of assumptions about your API implementation
 - The LoopBack framework creates an API path for each model that you define
 - By default, API operations map to actions on model objects
- You develop APIs faster by focusing on the business logic and data
 - When you define a model, the LoopBack framework automatically creates a predefined REST API with a full set of create, retrieve, update, and delete operations

[LoopBack models, properties, and relationships](#)

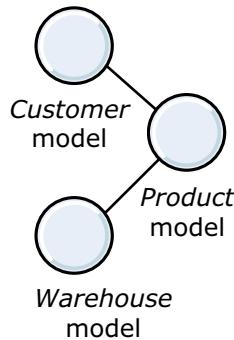
© Copyright IBM Corporation 2020

Figure 6-4. LoopBack: An open source Node.js API framework

LoopBack is an open source Node.js API framework for REST APIs.

LoopBack framework: Models, properties, relationships

- The model object represents the data and logic behind your API operations
 - Select the base class of the model from a list of built-in model classes, for example, *PersistedModel*
 - Use *Model* as the base class for models that use non-database connectors
- Properties represent a business data field
- Relationships define how API consumers create, retrieve, and modify models and model properties



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-5. LoopBack framework: Models, properties, relationships

When you define your model object, select the base class of the model from a list of built-in model classes.

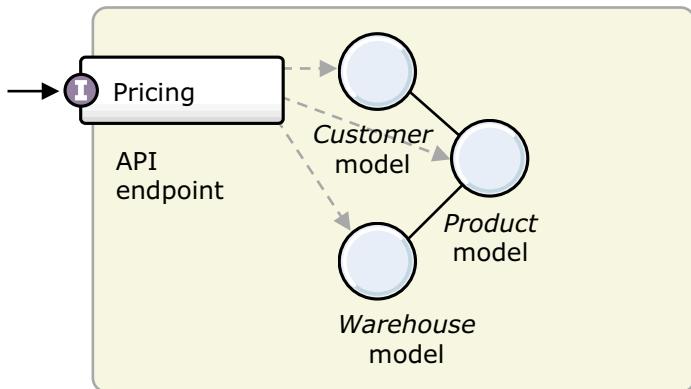
For models that map to databases, select *PersistedModel* as the base model class.

Use *Model* as the base class for models that connect with non-database connectors, such as the REST and SOAP connectors.

Each of the models that you define has a set of properties, and optionally relationships with other model objects. The lines between the model objects represent relationships between the models.

LoopBack framework: API mapped to models

- The framework automatically creates a set of **API operations** that give API consumers access to the model objects
 - To hide an API operation to all consumers, modify the model object code
 - To restrict API access to certain groups of consumers, apply authorization rules at the API Gateway through API Security Definitions



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-6. LoopBack framework: API mapped to models

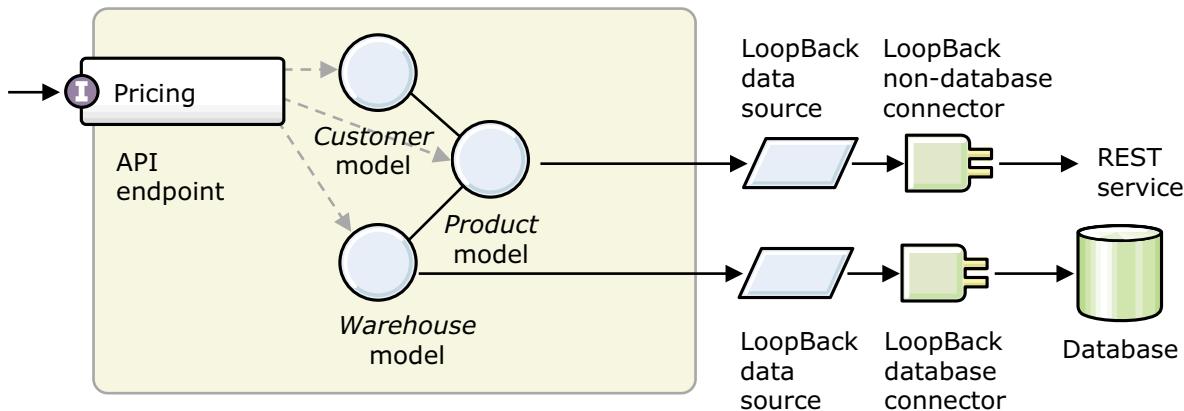
The LoopBack framework automatically adds create, retrieve, update, and delete API operations to each model object that belongs to the `PersistedModel` class.

To hide an API operation to all consumers, modify the model object code.

As a suggested practice, your API enforces role-based access control at the API gateway in an IBM API Connect architecture. You focus on implementing business logic within your LoopBack application.

LoopBack framework: Data persistence with connectors

- The framework also **retrieves** and **persists** the **properties** in the models to a data source that you define
 - To bind a data source to a database or data service, install and configure a **LoopBack connector**



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-7. LoopBack framework: Data persistence with connectors

A **LoopBack connector** is a Node module that connects model objects to sources of data outside of your LoopBack application. You can group LoopBack connectors into two categories: database and non-database connectors. Database connectors persist model data to databases.

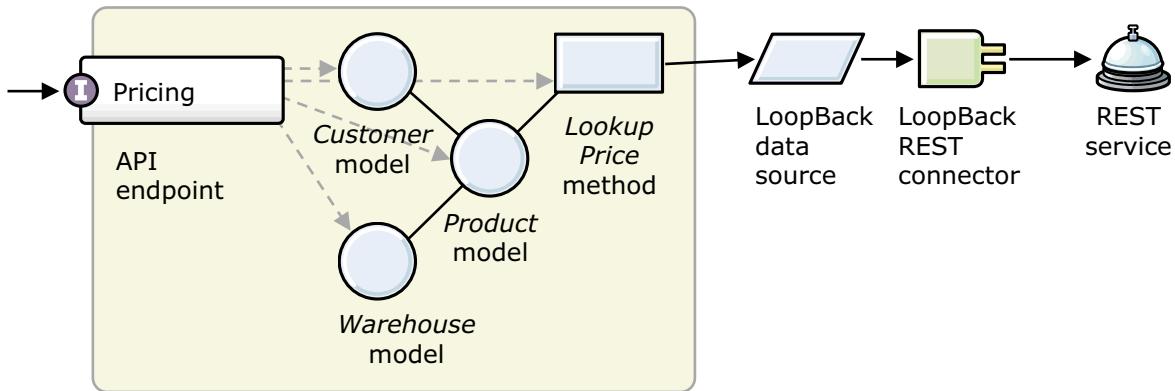
Non-database connectors do not support the persistence API: they call remote services and return data to a model object.

You configure a database connector in three steps.

1. You install a **LoopBack connector** for your type of database.
2. You define the connection information in a **LoopBack data source**.
3. You bind the model objects to the LoopBack data source.

LoopBack framework: Non-database connectors

- Not all LoopBack connectors persist model data to a data source
 - For example, the **REST** and **SOAP connectors** make remote web services available to your API implementation
- In this example, you install and configure the REST connector to call an existing REST service from your LoopBack application



[LoopBack models, properties, and relationships](#)

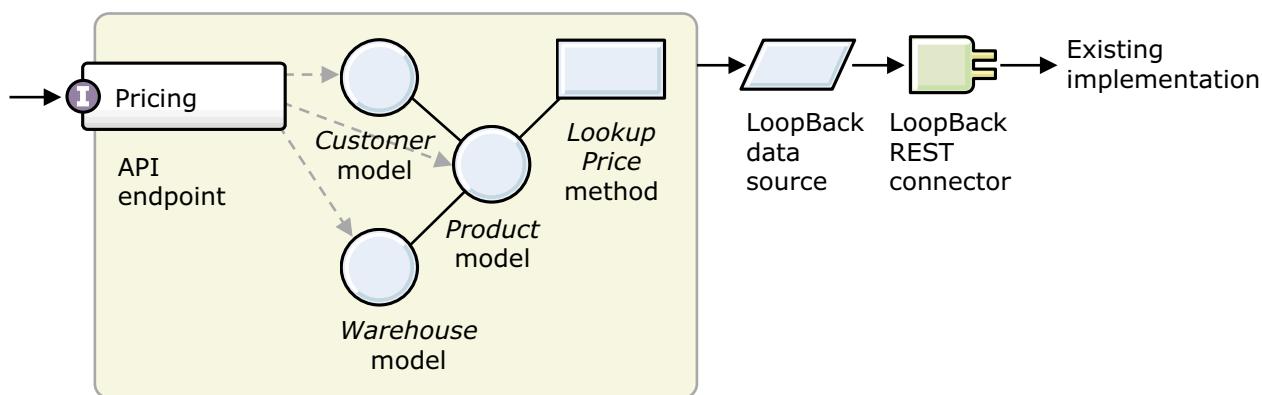
© Copyright IBM Corporation 2020

Figure 6-8. LoopBack framework: Non-database connectors

Non-database connectors do not support the persistence API: they call remote services and return data to a model object.

LoopBack framework: Remote methods and hooks

- The framework generates only a set of API operations that create, retrieve, update, and delete model and model properties
- To implement free-form API operations, create **remote methods** in the model
- To implement processing logic before and after API operations, create **remote hooks** in the model



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

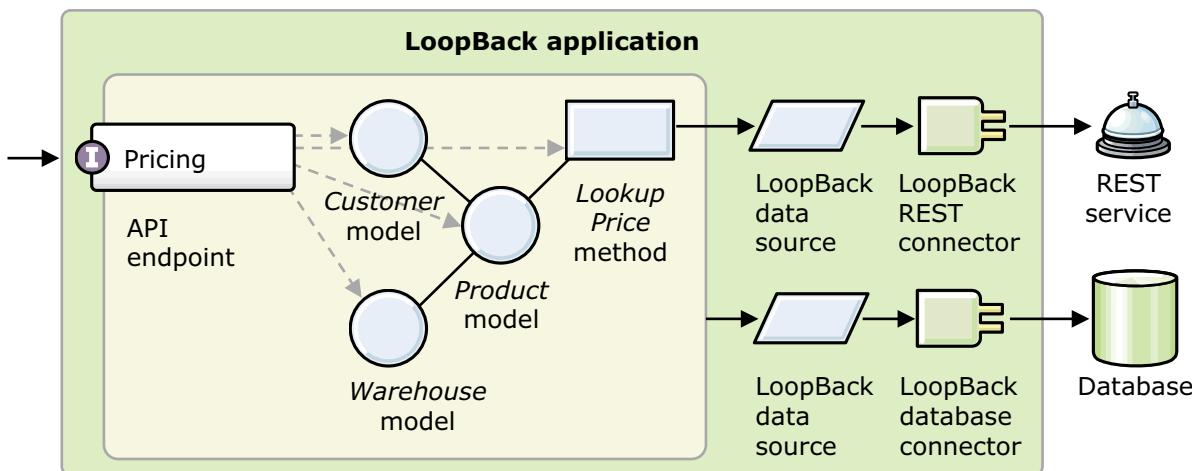
Figure 6-9. LoopBack framework: Remote methods and hooks

In this example, the *Lookup Price* function is an example of a **remote method**. It returns the price of a product based on business rules that you implement as a Node.js function. Since the *Lookup Price* function does not map to the database-centric create, retrieve, update, or delete operations, you must manually define this operation in the *price* model. An example of a remote method was shown in an earlier presentation unit.

Remote hooks are event handlers that run before and after an API operation. For example, you can define a remote hook that listens and logs each **lookup Price** request from the pricing application. In this scenario, you can write a remote hook that saves the price for a product after a client calls the *lookup price* API operation.

LoopBack framework: Packaging

- When you generate a LoopBack application with the `apic` command-line utility, you create a scaffold for an API implementation
 - You add the models, properties, and relationships
 - You install and configure the data sources and connectors
 - You implement the remote method in a Node.js function



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-10. LoopBack framework: Packaging

A LoopBack application is a Node.js application that is built around the structure of the LoopBack framework. The LoopBack framework maps a set of models to a preconfigured set of API operations. In this scheme, you define the structure of your models through configuration files. You implement custom behavior API operations as Node.js functions that are known as remote methods. You write code that intercepts API operations in Node.js functions that are known as remote hooks.

To store and retrieve model data, you configure a specific subset of LoopBack connectors that work with databases. The LoopBack framework automatically persists model data on your behalf.

To retrieve data from non-database sources, such as REST and SOAP operations, use non-database LoopBack connectors to access these resources.

The role of LoopBack in API Connect

- You configure two runtime components in an API Connect solution:
 - The **API Gateway** manages consumer access to system and interaction APIs
 - You provide the **implementation for APIs**, hosted on a server that the API Gateway can access
- A **LoopBack** application is one choice for **implementing APIs**
 - You publish the API definition as part of a product to the API Gateway
 - You deploy your LoopBack application on your own server, or on a Node.js container in IBM Bluemix
- The **IBM API Connect toolkit** is designed for application developers to configure and develop APIs and LoopBack applications

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-11. The role of LoopBack in API Connect

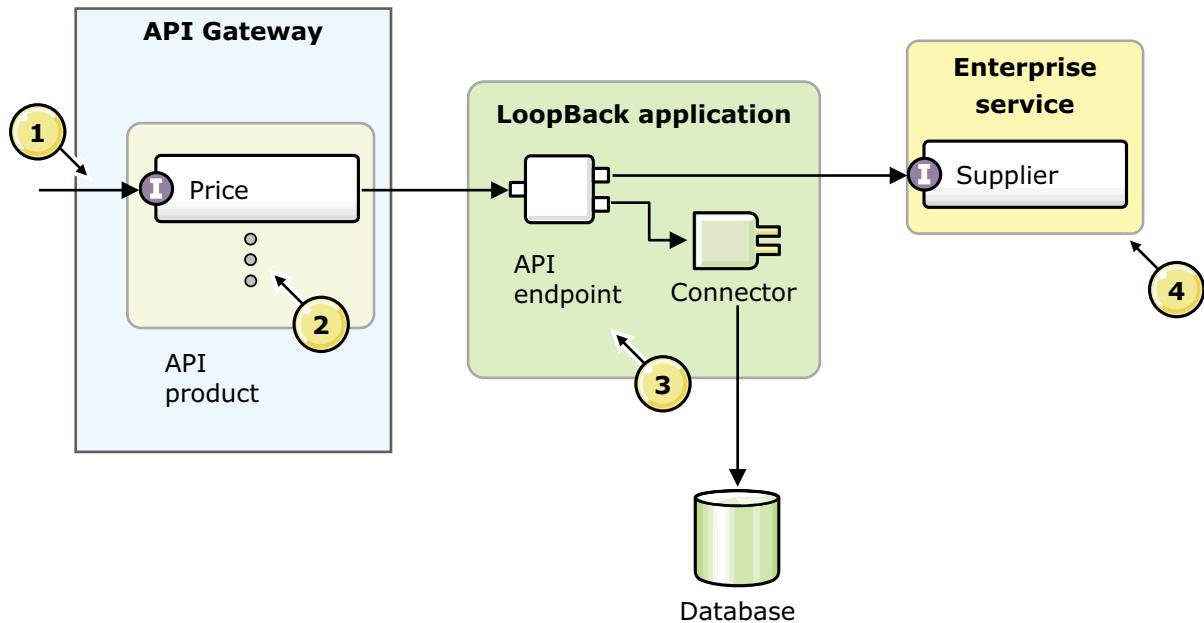
You configure two runtime components in an API Connect solution:

- The API gateway manages consumer access to system and interaction APIs.
You provide the implementation for APIs, hosted on a server that the API gateway can access.
- A LoopBack application is one choice for implementing APIs.
You publish the API definition as part of a product to the API gateway. You deploy your LoopBack application on your own server or as a Node.js application in a container runtime environment.

The IBM API Connect toolkit is designed for application developers to configure and develop APIs and LoopBack applications.

LoopBack applications in an API Connect solution

Core enterprise



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-12. LoopBack applications in an API Connect solution

1. In an IBM API Connect solution, application developers access your APIs through the API gateway. In this example, you published the Price API that you implemented in a LoopBack application.
2. The Price API is defined in an API product: a collection of APIs that you defined in the gateway. You learn more about the concept of API products and publishing in a later unit.
3. The API gateway delegates the API operation request to the API endpoint: a network address and port number for your LoopBack application. After you implemented and tested your LoopBack application, you must host the application on your own server, a container runtime, or on IBM Cloud.
4. The API that you implement in LoopBack can call other APIs, or existing APIs. In this scenario, your existing APIs are hosted in your enterprise architecture. The URL endpoint that the user calls on the gateway is different from the endpoint that the gateway calls the Loopback application.

For example: The user calls `https://<gateway_IP>/<org>/<catalog>/logistics/shipping`

The gateway calls: `http://<node_deployed_IP>:<port>/<method>?<arg1=value>&<arg2=value>`

Example: Note sample application

- In the `notes` LoopBack sample application, the **note** model stores a message
 - The message consists of two fields: `title` and `content`
- The LoopBack framework creates a set of create, retrieve, update, and delete operations for the **note** model
 - For example, you call `POST /notes` to store the title and content of a new note
 - Call `GET /notes` to retrieve all note objects on the server

[LoopBack models, properties, and relationships](#)

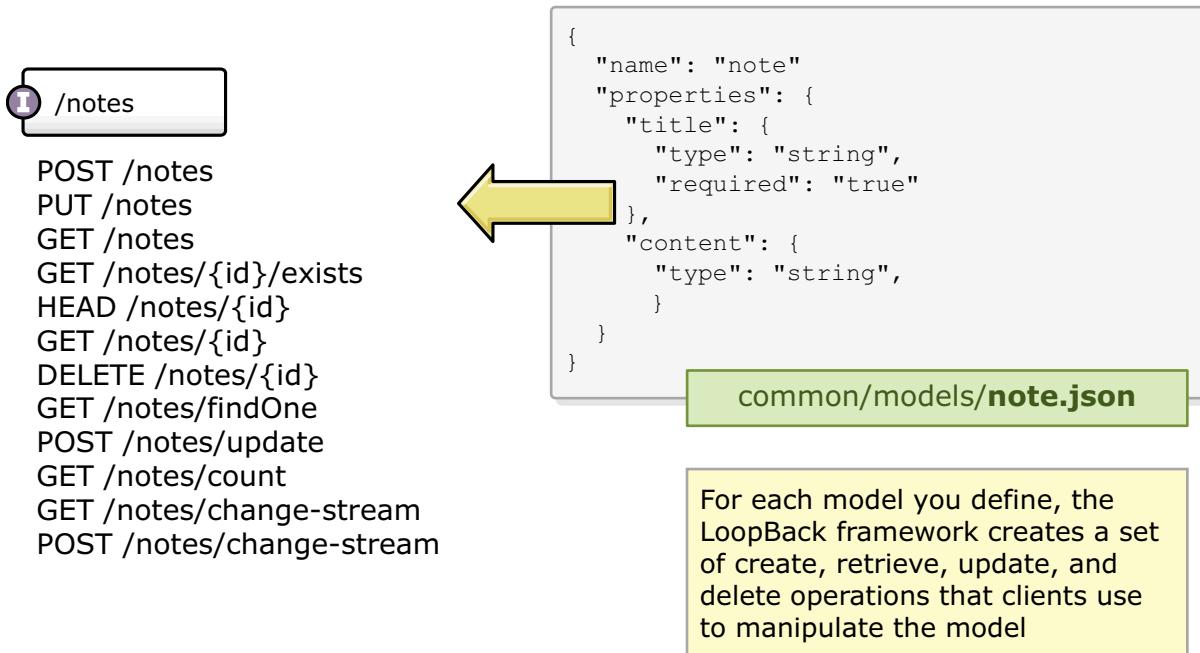
© Copyright IBM Corporation 2020

Figure 6-13. Example: Note sample application

To create a copy of the note sample LoopBack application, run `apic loopback` and select the **notes** application as a starting point.

The note sample includes an in-memory database that persists the message for the duration that the application runs.

Example: Note model



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-14. Example: Note model

In an earlier exercise, you generated the Note LoopBack application with the command-line tools. The `note.json` file declares the model name, properties, relationships, and other metadata. From this model definition file, the LoopBack framework creates a dozen data-centric API operations for you.

By default, the name of the API path is the plural form of the model name. In this example, the **notes** path represents the **note** model object.

- **POST /notes:** Create a note with the title and content as form parameters.
- **PUT /notes:** Update an existing note.
- **GET /notes:** Retrieve a list of all notes.
- **GET /notes/{id}/exists:** If a note with the specified identifier exists, return status code 200 OK.
- **GET /notes/{id}:** Retrieve the title and content for the specified note.
- **HEAD /notes/{id}:** Return the same value as **GET /notes/{id}**, but omit the message body in the response.
- **GET /notes/findOne:** Return the first note that matches the search filter parameters. Note: no ordering is assumed in the notes.
- **POST /notes/update:** Update an existing note. This operation is the same as **PUT /notes**.

- **GET /notes/count:** Return the total number of notes.
- **GET /notes/change-stream:** Return a running list of changes to any note in the application, as a stream of JSON objects.
- **POST /notes/change-stream:** Apply a set of changes to the notes in the application.

How to implement an API with the LoopBack framework

You follow five steps to implement an API with a LoopBack application

1. Generate a LoopBack application
2. Configure a data source to persist LoopBack model data
3. Define models, properties, and relationships
4. Create remote methods and hooks to add custom business logic
5. Review and test the API with the LoopBack Explorer web application

Next steps after you implement the API

- After you developed and tested your API, you *deploy* the LoopBack application to a Node.js runtime environment
- Make the API available on the gateway
 - Define a security policy in the API definition (interface) to secure client access to your API
 - Assemble message processing policies at the API Gateway to mediate and control API access
 - Add the API definition (interface) to a product in IBM API Connect to catalog the API
 - Publish the product to the API Gateway to make it accessible to API consumers

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-16. Next steps after you implement the API

6.2. LoopBack models, properties, and relationships

LoopBack models, properties, and relationships

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-17. LoopBack models, properties, and relationships

Topics

- LoopBack framework in API Connect
- ▶ LoopBack models, properties, and relationships

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-18. Topics

Steps: LoopBack application, models, properties, relations

In this topic, you focus on the following LoopBack application implementation steps:

- 1. Generate a LoopBack application**
- 2. Configure a data source to persist LoopBack model data**
- 3. Define models, properties, and relationships**
- 4. Create remote methods and hooks to add custom business logic**
- 5. Review and test the API with the API Explorer web application**

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-19. Steps: LoopBack application, models, properties, relations

In this topic, you focus on the following LoopBack application implementation steps.

- Generate a LoopBack application.
- Define models, properties, and relationships.

In many cases you define the data source before you create the model, so that you can designate the data source when you create the model. You can also create the model and separately bind to the data source as you see in a later example in this presentation.

The next presentation unit covers the creation of data sources.

Generate a LoopBack application with apic loopback

- To implement an API, you create a LoopBack application with the **apic 1b** command
 - A LoopBack application is a Node.js application that follows the LoopBack framework structure
- The utility creates a set of directories, scripts, and configuration files for a LoopBack application
 - The collection of these starter files and directories is known as an **application scaffold**
- You have four starting points for your LoopBack application:
 - An `api-server` that defines a LoopBack API server with local user authorization
 - An `empty-server` API does not define any configured models or data sources
 - The `hello-world` API defines a remote method that returns a greeting
 - The `notes` API retrieves and saves its information into an in-memory data source

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-20. Generate a LoopBack application with apic loopback

Example: Create a LoopBack application

```
$ apic lb
```

The terminal output shows the following steps:

1. The command `$ apic lb` is entered.
2. The application name is set to `inventory`.
3. The project directory is set to `inventory`.
4. The application type is selected as `empty-server`.

```
? What's the name of your application? inventory
? Enter name of the directory to contain the project: inventory
? What kind of application do you have in mind? (Use arrow keys)
> empty-server (An empty LoopBack API, without any configured models
or data sources)

hello-world (A project containing a controller, including a single
vanilla Message and a single remote method)

notes (A project containing a basic working example, including a
memory database)
```

LoopBack models, properties, and relationships

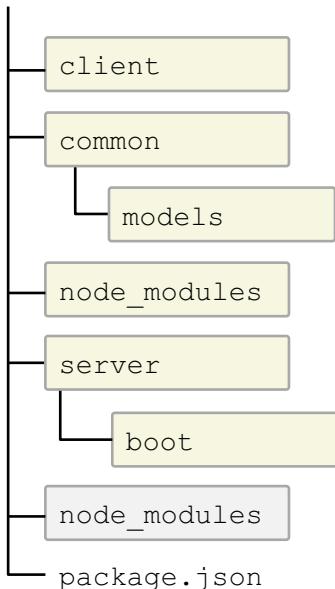
© Copyright IBM Corporation 2020

Figure 6-21. Example: Create a LoopBack application

1. In a terminal (Linux, Mac OS) or command prompt (Windows), run the `apic lb` command. The `apic lb` utility starts the LoopBack application generator. The LoopBack generator is based on the Yeoman open source project (<http://yeoman.io/>). The Yeoman project generates an application scaffold: a preconfigured set of directories, configuration files, and source code for the application framework that you choose.
2. The name of the application is the name of the current directory. If you change the application name, you enter the name of a new directory in the next step.
3. By default, the name of the directory that holds the application source code is the application name.
4. Select a starting point for the LoopBack application: an empty server with no models, the hello-world application, or the note application with a model object and in-memory data source.

You examine the files that the LoopBack generator creates in the next slide.

LoopBack application: Directory structure



- The LoopBack application structure contains four directories:
 - **client** stores the application front-end files
 - **common** stores the model classes and custom Node scripts for your application
 - **node_modules** stores the `node_modules` that are needed for the application to run
 - **server** stores the boot scripts and configuration files for the LoopBack framework
- In addition, the `package.json` manifest file describes the name, author, version, and package dependencies to the Node.js runtime
 - The node package manager saves local dependencies in the `node_modules` folder

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-22. LoopBack application: Directory structure

The diagram shows the directory structure for a generated Loopback application.

The `node_modules` directory is a standard location to store application-specific Node modules.

Earlier versions of Loopback also created a definitions directory that stored the definition for the API interface. To create an interface definition for the application, you must manually run the `apic lb export-api-def` command in this version of the API Connect Developer toolkit.

Example: Create a model and properties

The diagram shows a terminal session with numbered annotations:

- Annotation 1:** \$ apic lb model
- Annotation 2:** ? Enter the model name: **product**
- Annotation 3:** ? Select model's base class: **PersistedModel**
- Annotation 4:** ? Expose product via the REST API? **Yes**
- Annotation 5:** ? Custom plural form (used to build REST URL):
- Annotation 6:** ? Common model or server only? **Common**

Let's add some product properties now.
Enter an empty property name when done.

```
? Property name: name
    invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]:
```

Let's add another product property.
Enter an empty property name when done.

```
? Property name:
```

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-23. Example: Create a model and properties

1. In a terminal (Linux, Mac OS) or command prompt (Windows) where the API Connect Developer toolkit is installed, run the `apic lb model` command.
2. Enter a name and a base class for your custom model. Each base class has its own features and properties. For example, the **persisted model** base class works with the LoopBack database connector that you configure to persist model data.
3. The **Expose product via the REST API** option creates a preconfigured set of create, retrieve, update, and delete API operations for your custom model.
4. The **Custom plural form** setting overrides the spelling of the model in the API path. For example, the name of the model is **product**. The API path uses the plural form of the name: `/api/products/`
5. The **Common model or server only** settings asks where you want to place the model configuration and JavaScript code. Select **common** to create a directory that is named `models` in the `common` directory.
6. The **property** generator takes in four parameters: the `property name`, the `property type`, whether it is a `required` property, and a `default value`. To stop the LoopBack property generator, leave the property name blank and press **Enter**.

Where does LoopBack store model information?

- For each LoopBack model that you define, the generation tools creates two files:
 - The **model definition** defines the characteristics of your model, in JSON format
 - The **model script** defines custom logic for your model, such as a remote method, validation functions, and operation hooks

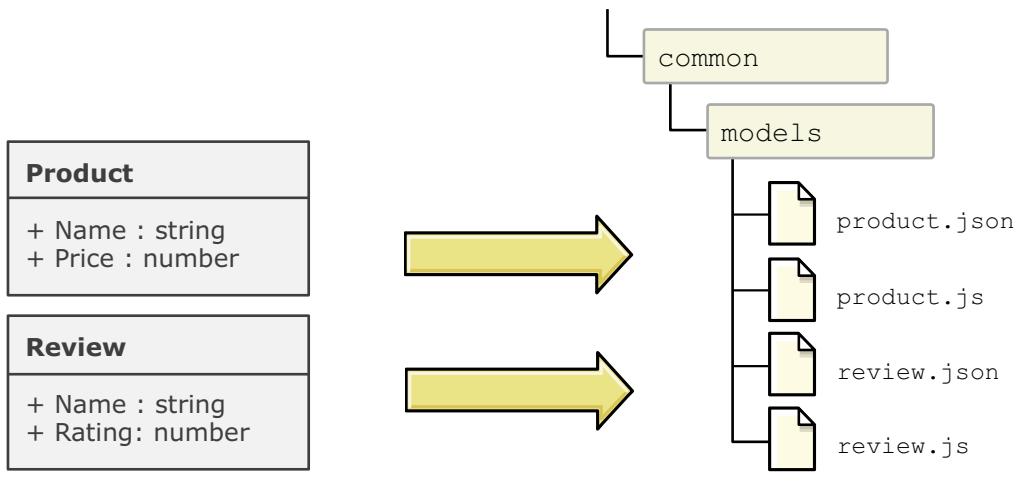
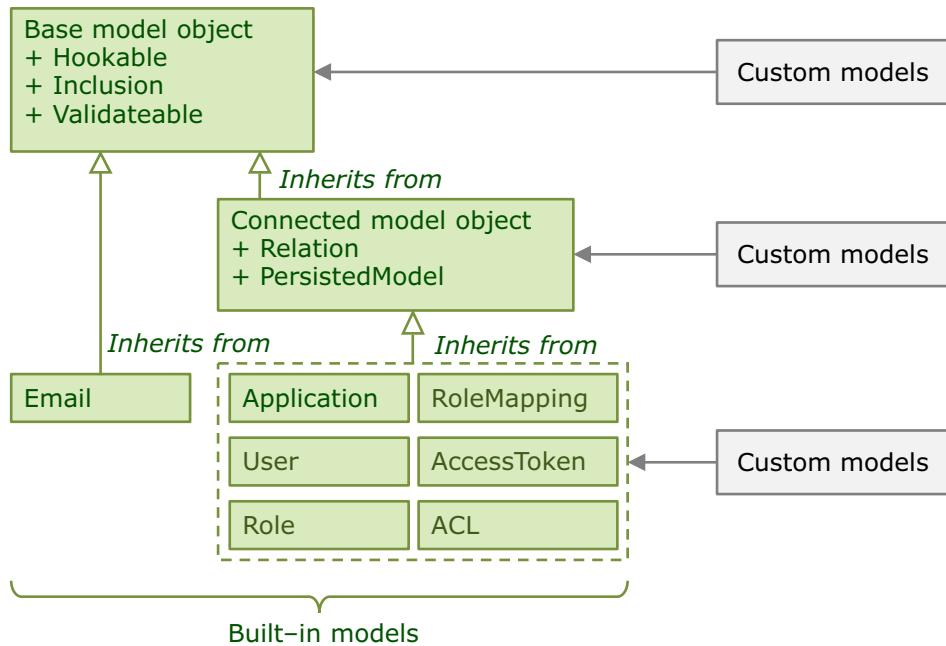


Figure 6-24. Where does LoopBack store model information?

The left side of the diagram depicts the two model objects in UML notation. The product model has two properties: a *name* and a *price*. The review model has a *name* in string format and a *rating* as a number.

The model script defines custom logic for the model. Examples of custom logic include remote methods, validation functions, and operation hooks.

LoopBack model inheritance



LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-25. LoopBack model inheritance

When you create a model object in your LoopBack application, you enter the base class for the model. Each base class has a set of features that you can extend in your own custom model.

The **model** class (listed as the Base model object) is the most generic type of model. This class supports remote hooks and validation methods. The **Email** connector is a built-in class that extends the model class.

The most common LoopBack base class is the **PersistedModel** class (listed as the Connected model object). The PersistedModel class inherits all of the features from the model class. In addition, it supports data persistence through database LoopBack connectors and model relationships.

LoopBack properties: Data types

Type	Description	Example
Array	JSON array	["one", 2, true]
Boolean	JSON Boolean	true
Buffer	Node.js Buffer object	new Buffer(42);
Date	JavaScript Date object	new Date("March 17, 2019 01:19:00");
GeoPoint	LoopBack GeoPoint object	new GeoPoint({lat: 43.849, lng: -79.338});
Number	JSON number	3.1415
Object	JSON object	{ "product": "Dayton", "price": "22.51"}
String	JSON string	"API Connect"
null	JSON null	null
any	Any type that is listed above	Any of: true, 123, "hello", ["one", 2, true]

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-26. LoopBack properties: Data types

This table lists the data types that you can specify for a model property or a parameter in a remote method. LoopBack supports JSON data types: array, boolean, number, object, string, and null. In addition, LoopBack supports three extra data types. The Node.js Buffer object is designed for remote methods that take a stream of data as input. The Date object stores the time and date in the standard JavaScript date format. The GeoPoint object stores one set of latitude and longitude map coordinates. Use GeoPoint to store a map location, or to calculate the distance between two places.

Example: Model definition file

```
{
  "name": "product",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "name": {
      "type": "string",
      "required": true
    },
    "price": {
      "type": "number"
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```

common/models/**product.json**

The **model definition** file defines the **characteristics** for your custom model

- In this example, you created a model that is named **product**
- The model inherits the features of the **Persisted Model** LoopBack base class

The **properties** section stores the model properties that you specified in the LoopBack property generator from the `apic create` command

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-27. Example: Model definition file

The **model definition** file captures the settings that you defined for your custom model. The model definition file is located in the /common/models directory subdirectory of your application. In this example, the name of the model is **product**, and the name of the model definition file is **product.json**. The LoopBack framework creates an instance of the base class, **PersistedModel**, as a template for your **product** model object.

The **idInjection** property indicates whether the LoopBack framework adds an **ID** property with a uniquely generated identifier.

The **validateUpsert** property indicates whether the LoopBack framework runs validation methods when the client calls a **create** or **update** API operation. You must define your custom validation methods in the model script file.

The **properties** section stores the model properties that you specified in the LoopBack property generator from the `apic create` command.

Example: Model script file

```
'use strict';

module.exports = function(Product) {
}
```

The **model script** file defines the **behavior** that you add to your model

common/models/product.js

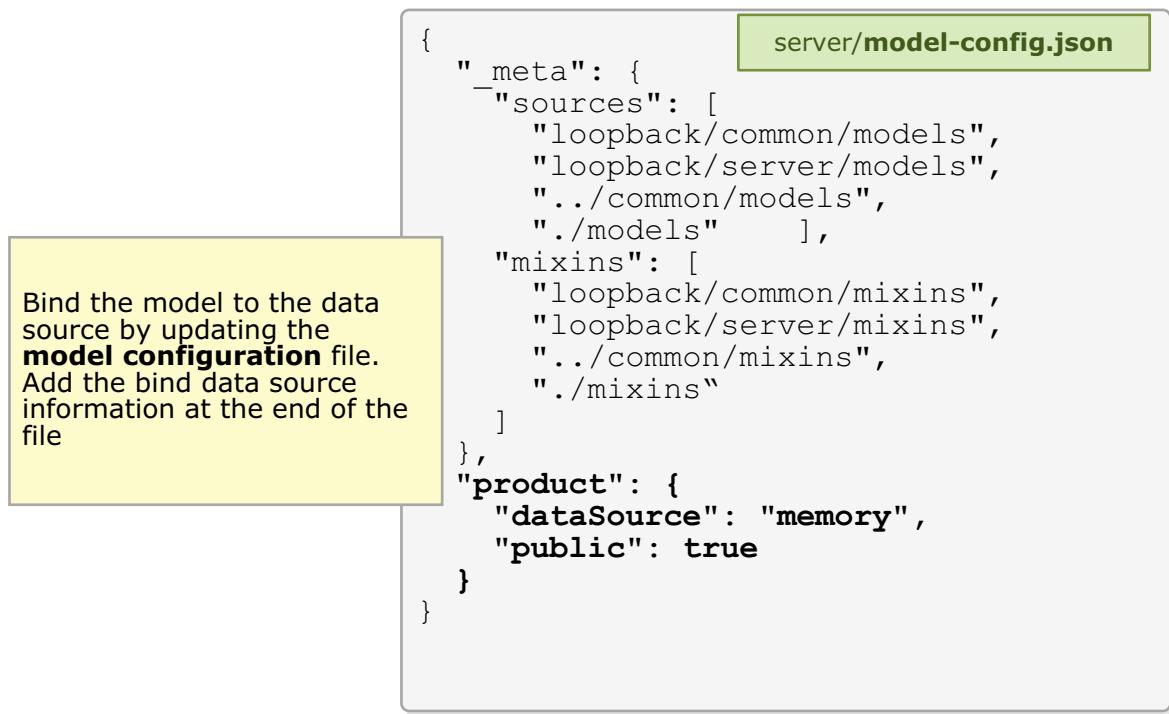
The code that you write in the model script extends or modifies behavior from the base class

- For example, the **product** model extends the **persisted model** base class
- Therefore, you do not need to write code to persist model data to a data source

Figure 6-28. Example: Model script file

The base class that you choose for your custom model already contains a set of properties and functions. You do not need to write custom code for the predefined create, retrieve, update, and delete API operations. You also do not need to write code that persists your model properties to a data source. The **persisted model** base class already implemented these features for you.

Example: Bind the model to a data source



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-29. Example: Bind the model to a data source

If you create the data source after you create the model, then you can bind the model to the data source manually by updating the model configuration file.

In this example, the in-built Loopback memory connector data source is added to the end of the model-config.json file.

How do you define a relationship between models?

- A **LoopBack model relation** is a relationship between model objects in a single LoopBack application
- When you define a relationship between models, the LoopBack framework adds a set of APIs to interact with each of the model instances
 - For example, you create a **one-to-many** relationship that is named **reviews** in the **review** model
 - You define the model relationship in `product.json`
 - Loopback adds a set of **search and query APIs** that retrieves product models that the warehouse owns



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-30. How do you define a relationship between models?

A **LoopBack model relation** is a relationship between model objects in a single LoopBack application.

When you define a relationship between models, the LoopBack framework adds a set of APIs to interact with each of the model instances.

In this example, you define a one-to-many relationship in the **product** model. The product has many reviews.

To save this relationship information, you create a relationship with the `apic` command-line utility. The command adds a “has many” relationship in the **product** model definition file, `product.json`.

The LoopBack framework adds a set of create, retrieve, update, and delete operations that API consumers use to modify the relationship information. For example, you can add a review to a product with the `POST /products/{productId}/reviews/` API operation.

Example: Create model relations

```
$ apic lb relation
```

? Select the model to create the relationship from:
> product
review

? Relation type:
> has many
 belongs to
 has many and belongs to
 has one

? Choose a model to create a relationship with:
 product
> review
 (other)

? Enter the property name for the relation:
reviews

The terminal output is annotated with five yellow circles containing numbers 1 through 5, each connected by a curved arrow pointing to a specific part of the command-line interaction:

- Step 1: Points to the first question mark and the first model selection (> product).
- Step 2: Points to the second question mark and the second model selection (> review).
- Step 3: Points to the third question mark and the relation type selection (> has many).
- Step 4: Points to the fourth question mark and the target model selection (> review).
- Step 5: Points to the fifth question mark and the property name entry (reviews).

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-31. Example: Create model relations

1. In a terminal (Linux, Mac OS) or command prompt (Windows), where the API Connect Developer toolkit is installed, run the `apic lb relation` command.
2. Select the model from which you want to create the relationship. In this example, you define a relationship in the **product** model.
3. Select a model relation type. The four relation types map to one-to-many, many-to-one, many-to-many, and one-to-one relationships.
4. Select the model to which you want to create the relationship. In this example, you define a relationship from the **product** to the **review** model.
5. Enter a name for the relationship. This name cannot be an existing property name in the model.

In this example, you stated that a **product has many reviews** in its inventory. The name of the relation is **reviews**.

Example: Relations in the model definition file

```
{  
  "name": "product",  
  ...  
  "relations": {  
    "reviews": {  
      "model": "review",  
      "type": "hasMany",  
      "foreignKey": "reviewId"  
    }  
  },  
  ...  
}
```

In the **model definition** file, review the **relations** section for a list of all relationships that start from this model

- In this example, you defined a model relation named **reviews**
- The product model **has many** reviews
- The way that you track the relationship is to store a collection of **reviewId** values

common/models/**product.json**

Figure 6-32. Example: Relations in the model definition file

Model relation types

- For example, you create the **product**, **review**, and **warehouse** models

Relation type	Example
Has many	A product has many reviews
Belongs to	A review belongs to exactly one product
Has and belongs to many	A warehouse stores many products , and a product belongs to many warehouses
Has one	A warehouse has exactly one product
Has many through	A warehouse stores many products <ul style="list-style-type: none"> ▪ Create a third model (a through-model) that maps warehouses to products

- For more information about relation types, see:
<https://loopback.io/doc/en/lb3/Creating-model-relations.html>

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-33. Model relation types

This slide gives examples on the five model relation types that you can model in LoopBack.

In this scenario, you create two models that are named **product** and **review**. The five model relationship types are as follows:

- Has many: A **product** has many **reviews**.
- Belongs to: A **review** belongs to exactly one **product**.
- Has and belongs to many: A **warehouse** stores many **products**, and a **product** belongs to many **warehouses**.
- Has one: A **warehouse** has exactly one **product**.
- Has many through: A **warehouse** stores many products. Create a third model (a through-model) that maps **warehouses** to **products**.

Unit summary

- Explain the relationship between the model and the API
- Explain the concepts of LoopBack models, properties, and relationships
- Explain the features that each model base class inherits
- Identify the property data types
- List the model relationship types
- Define models, properties, and relationships in the API Connect Developer toolkit apic utility

[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-34. Unit summary

Review questions

1. Which statement best describes the LoopBack framework?
 - A. LoopBack defines how the API gateway processes request and response messages.
 - B. LoopBack maps data-centric API operations to models that you create.
 - C. LoopBack creates databases that store model data.
 - D. LoopBack enforces an API development lifecycle.

2. In the product and warehouse scenario, how do you retrieve a list of products that a particular warehouse stores?
 - A. Develop a remote method that searches the list of warehouses.
 - B. Develop a remote hook that filters products by warehouses.
 - C. Call a GET method on product with a filter by warehouses.
 - D. Create a through-model that tracks the product to warehouse mapping.



[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2020

Figure 6-35. Review questions

Write your answers here:

1.

Review answers

1. Which statement best describes the LoopBack framework?
 - A. LoopBack defines how the API gateway processes request and response messages.
 - B. LoopBack maps data-centric API operations to models that you create.
 - C. LoopBack creates databases that store model data.
 - D. LoopBack enforces an API development lifecycle.

The answer is B.

2. In the product and warehouse scenario, how do you retrieve a list of products that a particular warehouse stores?
 - A. Develop a remote method that searches the list of warehouses.
 - B. Develop a remote hook that filters products by warehouses.
 - C. Call a GET method on product with a filter by warehouses.
 - D. Create a through-model that tracks the product to warehouse mapping.

The answer is C.

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2020

Figure 6-36. Review answers



Unit 7. Defining data sources with connectors

Estimated time

01:00

Overview

This unit explores how LoopBack data sources retrieve and save model data from data stores. You learn how to install LoopBack connectors in your application, and how to map model objects to data sources.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Explain the role of a LoopBack data source
- Identify the role of database and non-database connectors
- Install a LoopBack connector
- Explain how to map data sources to models
- Generate an application that uses the built-in memory connector
- Build model instances from unstructured data

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-1. Unit objectives

7.1. LoopBack data sources

LoopBack data sources

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-2. LoopBack data sources

Topics

▶ LoopBack data sources

- Generating a LoopBack application that uses an in-memory connector
- Generating LoopBack models from discovery and introspection

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-3. Topics

Steps: LoopBack data sources

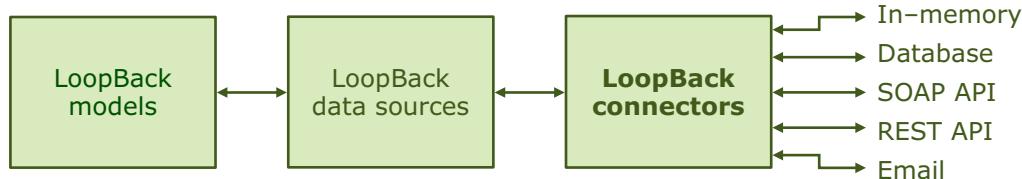
In this topic, you focus on the following LoopBack application implementation step:

1. Generate a LoopBack application
2. **Configure a data source** to persist LoopBack model data
3. Define models, properties, and relationships
4. Create remote methods and hooks to add custom business logic
5. Review and test the API with the API Explorer web application

Figure 7-4. Steps: LoopBack data sources

In this topic, you focus on the following LoopBack application implementation step: **Configure a data source** to persist LoopBack model data.

What is a LoopBack connector?



- LoopBack connectors **connect models to external sources of data** through create, retrieve, update, and delete functions
- LoopBack also generalizes other back-end services, such as REST APIs, SOAP web services, and storage services, as data sources
- Connectors implement the data exchange logic for data sources
 - In general, applications do not use connectors directly
 - Your model accesses data sources through the `DataSource` and `PersistedModel` APIs

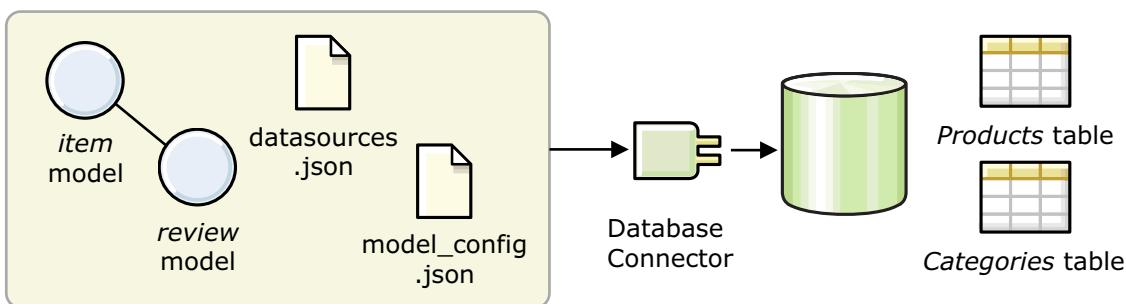
Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-5. What is a LoopBack connector?

How do you persist model data with connectors?

- Before you create models for your LoopBack application, **set up a data source** to persist the model data
- The most common type of LoopBack model is a **persisted model**
 - The LoopBack framework keeps the model data consistent with the data source through a **LoopBack connector**
 - You define the connection details to the data source in the `datasources.json` configuration file
 - You map your models to a data source in the `model-config.json` configuration file



Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-6. How do you persist model data with connectors?

The diagram assumes that you are mapping your model objects to a relational database with a LoopBack connector. If you configure a non-relational database, such as IBM Cloudant, you map your models to **databases** in the **data source**. Each **model instance** maps to a **document** in the **database**.

Database connectors

- These LoopBack connectors implement create, retrieve, update, and delete operations to models that extend the PersistedModel interface
 - Cassandra
 - IBM Cloudant
 - DashDB
 - DB2, DB2 for iSeries, DB2 for z/OS
 - Informix
 - MongoDB
 - MySQL
 - Oracle
 - PostgreSQL
 - Redis
 - SQL Server
 - SQLite3
 - z/OS Connect Enterprise Edition connector
- Review the documentation for an up-to-date list of database connectors: <https://loopback.io/doc/en/lb3/Database-connectors.html>

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-7. Database connectors

The list of LoopBack connectors is constantly updated. Review the documentation for the current list of connectors, and detailed instructions on how to configure each connector.

Memory connector

- Built-in memory connector is a persistent data source for development and testing
 - Test your application without connecting to a persistent data source such as a database
 - Does not require you to set up a database
- Built-in memory connector supports:
 - Create, read, update, and delete operations
 - Standard queries
- You can persist data between application restarts by using the `file` property with the built-in memory connector

[Defining data sources with connectors](#)

© Copyright IBM Corporation 2020

Figure 7-8. Memory connector

With the LoopBack built-in memory connector, you can test your application without connecting to an actual persistent data source such as a database. Although the memory connector is well tested, it is not suitable for production.

Built-in memory connector supports: Create, read, update, and delete operations and standard queries.

You can persist data between application restarts by using the `file` property with the built-in memory connector.

Example: Create a memory data source

```
$ apic lb datasource
? Enter the data-source name: memoryds
? Select the connector for memoryds:
  In-memory db (supported by StrongLoop)
  IBM DB2 (supported by StrongLoop)
  IBM Cloudant DB (supported by StrongLoop)
  MongoDB (supported by StrongLoop)

Connector-specific configuration:
? window.localStorage key to use for persistence (browser only):
? Full path to file for persistence (server only):
  ./server/data/memory.json
```

Figure 7-9. Example: Create a memory data source

1. To define a data source, run the `apic lb datasource` command from the terminal (Linux, Mac OS) or command prompt (Windows).
2. Specify a name for the LoopBack data source.
3. Select a **LoopBack connector** from the list. In this example, you select the **in-memory** database connector. This connector is part of the LoopBack framework; you do not need to download another Node module.
4. The remaining questions are specific to the in-memory connector. LoopBack applications can run on the client side as a web browser application, and as a server-side resource. Since you are building an interaction API, you do not use the HTML local storage feature for client-side storage. Leave the `window.localStorage` key blank.
5. In the second question, you can persist in-memory data to a JSON document. By saving the in-memory database, you save your model data when you stop and start your LoopBack application.

Example: Data sources configuration file

```
{
  "memoryds": {
    "name": "memoryds",
    "connector": "memory",
    "localStorage": "",
    "file": "./server/data/memory.json"
  }
}
```

The **data sources** configuration file defines all data sources that you defined in your LoopBack application

In this example, you defined a data source named **memoryds**
 ▪ It uses the **in-memory** connector
 You set a file on your local disk to persist model data from memory

server/**datasources.json**

Figure 7-10. Example: Data sources configuration file

By default, the memory connector does not persist model data. However, you can specify a disk location to save model data. When you specify the **file** property, the connector saves your model data in a JSON document. When you restart your LoopBack application, the memory connector restores the data from the JSON file.

Example: Model configuration file

The **model-config** settings file maps **models** to **data sources**

- The LoopBack framework uses the `_meta` section
- Leave this section of the file unchanged

- In the remaining sections of the file, each JSON object stores the name of a **model** object
- For each model, the **data source** declares which data source LoopBack uses to retrieve data
- The **public** setting determines whether LoopBack adds API operations for the model object

```
{
  "_meta": {
    "sources": [
      "loopback/common/models",
      "loopback/server/models",
      "../common/models",
      "./models" ],
    "mixins": [
      "loopback/common/mixins",
      "loopback/server/mixins",
      "../common/mixins",
      "./mixins" ]
  },
  "item": {
    "dataSource": "mysql-connection",
    "public": true
  },
  "review": {
    "dataSource": "mongodb-connection",
    "public": true
  }
}
```

server/model-config.json

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-11. Example: Model configuration file

Setting the meta section aside, the model configuration file is a list of models in your LoopBack application. For each model, you define the data source from which LoopBack retrieves model data. If you configured a database data source, the LoopBack connector also persists data from your models to the data source.

The **public** setting determines whether LoopBack exposes a set of create, retrieve, update, and delete operations for the model. If the public property is **false**, your API consumers cannot access the properties from that model. However, your LoopBack application can access the model programmatically.

Non-database connectors

- Beyond databases, LoopBack supports a set of connectors that implement specific ways to access remote systems
 - Email connector
 - JSON RPC connector
 - MQ Light connector
 - Push connector
 - Remote connector
 - REST connector
 - SOAP connector
 - Storage connector
 - Swagger connector
- Models that are attached to non-database sources serve as connectors: the model classes contain only methods
 - Since non-database connectors do not support create, retrieve, update, and delete operations, these models usually do not define properties
- For more information about each non-database connector type, see:
<https://loopback.io/doc/en/lb3/Other-connectors.html>

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-12. Non-database connectors

For example, the REST connector delegates calls to REST APIs while the push connector integrates with iOS and Android push notification services.

7.2. Generating a LoopBack application that uses an in-memory connector

Generating a LoopBack application that uses an in-memory connector

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-13. Generating a LoopBack application that uses an *in-memory* connector

Topics

- LoopBack data sources
- ▶ Generating a LoopBack application that uses an in-memory connector
- Generating LoopBack models from discovery and introspection

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-14. Topics

Create a LoopBack local file application: example

In this scenario, you implement an API with the built-in LoopBack file-based connector

1. Generate a LoopBack application with the API Connect toolkit
2. Define an in-memory data source connector
3. Configure the data source that supplies information to your application
4. Create the models and properties in the LoopBack application
5. Change the default behavior for the id property so that it does not inject the id
6. Start the Loopback application and the LoopBack Explorer
7. Review the application in the LoopBack Explorer
8. Test the operation in the Explorer
9. Review the responses from calling the API with the LoopBack REST connector

[Defining data sources with connectors](#)

© Copyright IBM Corporation 2020

Figure 7-15. Create a LoopBack local file application: example

In this topic, you review an example on how to implement an API with the IBM API Connect Toolkit. The example persists data to a local file.

Students are encouraged to create this LoopBack example on the remote lab platform.



Information

If you do not want to build the example yourself, you can run it from the /home/localuser/lab_files/demos/product directory.

This scenario mimics the creation of the data records that you saw earlier in the <https://pricesample.mybluemix.net/price/> application. The application is built with LoopBack and deployed to the IBM Cloud.

Step 1: Generate a LoopBack application

The IBM API Connect toolkit is installed on the student lab environment. Use the command-line LoopBack generation tool to create the application scaffold

1. Open a terminal
2. Run the LoopBack application generator

```
$ apic lb
? What's the name of your application? product
? Enter name of the directory to contain the project: product
    Info change the working directory to inventory
? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind?
    empty-server API, without any configured models or data
sources
```

3. Change directory into your LoopBack application

```
$ cd product
```

Figure 7-16. Step 1: Generate a LoopBack application

Step 1: Generate a LoopBack application.

After you install the IBM API Connect toolkit, use the command line LoopBack generation tool to create the application scaffold.

1. Open a terminal (Linux).
2. Run the LoopBack application generator.
3. Change directory into your LoopBack application.

The `apic lb` command creates the directory structure and a set of configuration files in a directory. By default, the directory is the same name as your application name.

You must complete the rest of the steps in the LoopBack directory. If you call the `apic` commands in another directory, they do not work.

Step 2: Define a data source

Before you define your model objects, configure the data sources that supply data to your models. Ensure that you are in the product directory

1. Create a LoopBack data source

```
$ apic lb datasource
? Enter the data-source name: myfile
? Select the connector for myfile:
    In-memory db (supported by StrongLoop)
? window.localStorage key to use for persistence (browser only):

? Full path for persistence (server only): ./server/memory-
db/data.json
```

2. Review and edit the data source settings

```
$ cat server/datasources.json
```

Figure 7-17. Step 2: Define a data source

Step 2: Define a data source.

Before you define your model objects, configure the data sources that supply data to your models.

1. Create a LoopBack data source.
2. Review the data source settings.

Step 3: Configure the data source persistence file

- In this example, you create the folder and the empty `data.json` data file under the product/server/ directory

Create the directory:

```
$home/localuser/product> mkdir server/memory-db
```

Create the file that is used for persistence:

```
$home/localuser/product> touch server/memory-db/data.json
```

- The layout of the `data.json` file when data is added is shown:

```
{
  "ids": {
    "product": 4  },
  "models": {
    "product": {
      "1": "{\"product-id\": \"1\", \"price\":100}",
      "2": "{\"product-id\": \"2\", \"price\":200}",
      "3": "{\"product-id\": \"3\", \"price\":300.15}"
    }
  }
}
```

server/memory-db/data.json

Figure 7-18. Step 3: Configure the data source persistence file

Step 3: Create the folder and empty file that is the target that is specified in the persistence path in step 2.

If you want to populate the empty `data.json` file, this operation is done by calling the POST method in a later step. The remainder of the steps in the example assumes that you are starting with an empty data file.



Information

If you already have data that you want to populate in the `data.json` file, then copy the `/home/localuser/lab_files/demos/product/memory-db/` folder to the `/home/localuser/product/server/` folder.

The data types and properties must match the types and properties that you specify when you create the model and properties in step 4.

Step 4: Create the models and properties

- Create the LoopBack model and properties

```
$ apic lb model
? Enter the model name: product
? Select the data-source to attach undefined to:
  myfile
? Select model's base class PersistedModel
? Expose product via the REST API? Yes
? Custom plural form (used to build REST URL):
? Common model or server only? Common

Let's add some product properties now.
Enter an empty property name when done.
? Property name: product-id
? Property type: string
? Required: Y
? Default value:
Let's add another product property.
Enter an empty property name when done.
? Property name: price
? Property type: number
? Required: N
? Default value: 0
Enter an empty property name when done.
```

Figure 7-19. Step 4: Create the models and properties

Step 4: Create models, properties, and relationships.

- In a LoopBack application, model objects represent business data.
- The LoopBack framework retrieves and persists data with data sources.
- The LoopBack framework creates a set of API operations that API consumers use to access model data.

Step 5: Change the default id injection

- Edit the file common/models/product.json

```
{
  "name": "product",
  "base": "PersistedModel",
  "idInjection": false,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "product-id": {
      "type": "string",
      "required": true,
      "id": true,
      "generated": false
    },
    "price": {
      "type": "number",
      "default": 0
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-20. Step 5: Change the default id injection

Step 5: Change the default behavior for the id property so that it does not inject the id.

Open the file with the editor: gedit common/models/product.json

Change the line with idInjection from true to false:

"idInjection": false,

Also, add the lines to the product-id properties:

```
"id": true,
"generated": false
```

Save the change.



Information

You need only to make these changes if you want to manually add the identifier key, otherwise LoopBack automatically generates a “id” each time a product is added.

Step 6: Start the LoopBack application and Explorer

- To test your API, you start these components:
 - The **LoopBack application** implements the API
 - The **LoopBack Explorer** browser-based test function

- Start the LoopBack application from the product directory:
 - From the terminal type:

```
$ cd ~/product
$ npm start
> product@1.0.0 start/home/localuser/product
> node .
Web server listening at http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- The LoopBack NodeJS application and the LoopBack Explorer start
- Copy the URL with the LoopBack Explorer value into the address in a browser session

Figure 7-21. Step 6: Start the LoopBack application and Explorer

Step 6: Start the LoopBack application and the Loopback Explorer.

Start the LoopBack application by typing “npm start” in the folder where the LoopBack application was generated.

The LoopBack application starts and the LoopBack Explorer also starts automatically.

Copy the URL for the Explorer into the address area of your browser.



Step 7: Review the application in the Loopback Explorer

- Use the LoopBack Explorer as a test client for your interaction APIs
- Click **Show/Hide** or **List Operations** in the Explorer to view the operations
 - The list of operations for the product application is displayed on the page in the browser
 - To insert a product, select the `POST /products` operation

The screenshot shows the LoopBack API Explorer interface. At the top, it says "LoopBack API Explorer" and "Token Not Set" with a "Set Access Token" button. Below that, there's a search bar with the placeholder "product". Under the search bar, the word "product" is shown in bold. Then, there's a list of operations for the "product" model:

- PATCH /products**: Patch an existing model instance or insert a new one into the data source.
- GET /products**: Find all instances of the model matched by filter from the data source.
- PUT /products**: Replace an existing model instance or insert a new one into the data source.
- POST /products**: Create a new instance of the model and persist it into the data source.

The "POST /products" operation is highlighted with a red border.

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-22. Step 7: Review the application in the Loopback Explorer

Step 7: Review the application in the **LoopBack Explorer**.

The LoopBack Explorer is the integral test client for LoopBack applications.

Run the LoopBack Explorer by typing the address in the browser. The URL for the Explorer is displayed in the terminal that runs the application.

Click either the Show/Hide or the List Operations in the Explorer to display the operations for the application.

Select the operation that you want to test. In this example, the operation is named POST /products.

Step 8: Test the operation in the Loopback Explorer

- Select the POST /products operation.
Then, click **Try it out**

The screenshot shows the Loopback Explorer interface for testing API operations. The operation selected is **POST /products**. The response class is **Status 200**, and the request was successful. The model example value is:

```
{
  "product-id": "string",
  "price": 0
}
```

The response content type is **application/json**. The parameters section shows a parameter named **data** with a value:

```
{
  "product-id": "1",
  "price": 100.01
}
```

The parameter description is **Model instance data** and the parameter type is **body**. The data type is also **Model**. The parameter content type is **application/json**. A large arrow points from the parameter value area to the response example area.

The response example value is:

```
{
  "product-id": "string",
  "price": 0
}
```

At the bottom, there is a **Try it out!** button.

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-23. Step 8: Test the operation in the Loopback Explorer

Step 8: Test the API operation with the **LoopBack Explorer**.

When you select the operation, the page displays example responses and the required request parameters.

Select the POST /products operation.

Click inside the body of the example values. The values are copied into the parameter data area. Change the values to your input values.

Then, click **Try it out**.

Step 9: Review the results

- The product is added successfully
- Review the responses from calling the API with the LoopBack in-memory connector and a local file

The screenshot shows the IBM Cloud API Explorer interface. It displays a successful POST request to the endpoint `http://localhost:3000/api/products`. The request body contains the JSON object `{"product-id": "1", "price": 100.01}`. The response code is 200, and the response body is identical to the request body.

```

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ "product-id": "1", "price": 100.01 }' 'http://localhost:3000/api/products'

```

Request URL: `http://localhost:3000/api/products`

Response Body:

```
{
  "product-id": "1",
  "price": 100.01
}
```

Response Code: 200

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-24. Step 9: Review the results

Step 9: Review the response message in the Explorer.

A response code 200 and the response body are returned in the Explorer.

You successfully called the API from the LoopBack application with the in-memory connector and returned the file data.



Optional

Open the `https://pricesample.mybluemix.net/price/` application in a separate browser window. Run the POST /products operation to create the same data values for the product. When finished, call the GET /products operation in LoopBack. Then, review the data.json file that you created earlier. You replicated the cloud application as a local Node.js application.

7.3. Generating LoopBack models from discovery and introspection

Generating LoopBack models from discovery and introspection

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-25. Generating LoopBack models from discovery and introspection

Topics

- LoopBack data sources
- Generating a LoopBack application that uses an in-memory connector
- ▶ Generating LoopBack models from discovery and introspection

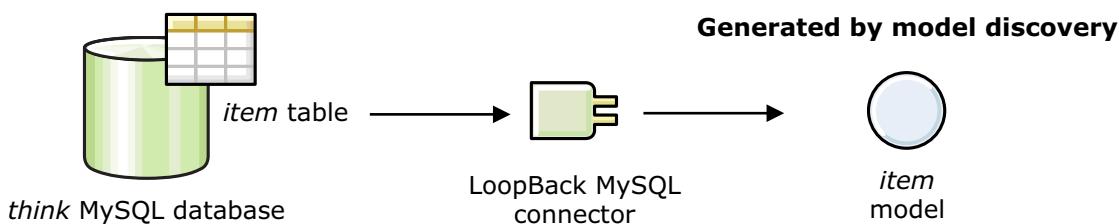
Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-26. Topics

Model discovery and introspection

- You learned how to define LoopBack models and properties that correspond to a database schema
 - However, defining properties in apic toolkit is a repetitive and error-prone process
- The **model discovery** feature automates the process for creating LoopBack models and properties from relational databases
- The **model introspection** feature generates a model instance and properties from a document in a non-relational database



Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-27. Model discovery and introspection

In the exercise case study, you defined the LoopBack model and properties with the API Connect Developer toolkit. However, this process is time-consuming and open to transcription error: you might make a mistake in the editor.

The LoopBack connector architecture provides two features that automate the process of creating LoopBack models. For relational databases, **model discovery** scans through the structure of database tables and generates a set of models. For non-relational databases, **model introspection** generates LoopBack objects for a particular document in the database.

You cannot run model discovery in a non-relational database. By their definition, this type of database holds unstructured data. However, you can instantiate a model object with **model** introspection, without defining a model configuration file beforehand.

Generate models from relational databases

- The **model discovery** feature builds model objects from relational databases
 - Create a LoopBack model from a database table
 - Define properties in the model based on the table columns
- The following data sources support model discovery:
 - Cassandra
 - MySQL
 - Oracle
 - PostgreSQL
 - SQL server connector
 - DB2
 - Dash DB
 - DB2 for z/OS
- For more information, see <https://loopback.io/doc/en/lb3/Discovering-models-from-relational-databases.html>

[Defining data sources with connectors](#)

© Copyright IBM Corporation 2020

Figure 7-28. Generate models from relational databases

The model discovery feature builds model objects from relational databases. You can create a LoopBack model from a database table. Within each table, you can define properties in the model based on the table columns.

The following data sources support model discovery in LoopBack version 3:

Cassandra

MySQL

Oracle

PostgreSQL

SQL server connector

DB2

Dash DBDB2 for z/OS

For more information, see

<https://loopback.io/doc/en/lb3/Discovering-models-from-relational-databases.html>



Information

Previous versions of the API Designer data source editor support these two features:

The **Discover Models** feature scans database tables, and generates LoopBack models based on the schema.

The **Update Schema** feature works in the reverse direction: it updates the database schema based on the LoopBack model and property structure.

These features were not integrated into the API Connect 2018.4.1.1 version of the API Designer at the release date of this course.

Instead, this course uses the command-line interface with LoopBack, so the API Designer model discovery feature is not covered in this course.

Create model instances from unstructured data

- Certain models return unstructured data in the response:
 - **Document-centric (NoSQL) databases** do not enforce a set data structure for all records in the database
 - **REST** and **SOAP** services return arbitrary JSON and XML data in the response message payload
- Model introspection generates a single model object instance from a document
- To create a LoopBack model instance from introspection, call the `buildModelFromInstance()` function from select data sources:
 - MongoDB data sources
 - REST data sources
 - SOAP data sources

Figure 7-29. Create model instances from unstructured data

Unlike a relational database, document-centric databases do not define a table structure. MongoDB is a document-centric, NoSQL, database that stores a collection of documents. Each document can contain an arbitrary number of fields.

You cannot “discover” a data structure from these types of data sources because the data itself is unstructured: each “record” can have a different data structure. However, you can create one single model instance from arbitrary data, and persist the model in the data source.

The `buildModelFromInstance` command takes a JSON document as a template to build a model object. With the model object, you can persist it in any data source. The following slides explain how to programmatically create a LoopBack model with the “`buildModelFromInstance`” function, and persist it into a data source.

Example: Create a model from introspection (1 of 2)

```
module.exports = function(server, callback) {
  var ds = server.dataSources.memoryds;

  ds.once('connected', function() {

    // Define a JSON document with sample data
    var article = {
      title: 'Create, secure, and publish APIs with IBM API Connect',
      author: 'anonymous',
      address: {
        street: '123 Main Street',
        city: 'Parksville',
        region: 'British Columbia',
        postcode: 'V7J 3R1',
        country: 'Canada'
      },
      email: [
        {label: 'work', id: 'poorjoe@example.com'}
      ],
      tags: ['ibm', 'cloud', 'apiconnect']
    }
  });
}
```

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-30. Example: Create a model from introspection (1 of 2)

To run the model introspection demonstration, create an “empty-server” LoopBack application with API Connect Toolkit. Define an in-memory data source with the name: **memoryds**

Create a boot script with the contents of the following two slides. You can find a copy of this demonstration in the `/home/student/lab_files/demos/introspection` directory.

In this example, the boot script takes a sample JSON document. The goal of this script is to create a LoopBack model based on the structure of this document. You can extend this example by retrieving any arbitrary JSON document from a NoSQL database, or REST service.



Information

The following data sources support instance introspection:

- MongoDB data sources
- REST data sources
- SOAP data sources

The example uses the built-in memory connector that does not officially support introspection.

Example: Create a model from introspection (2 of 2)

```
// Create a persisted model named 'Article', based on the
// structure of the 'article' JSON document.
var Article =
  ds.buildModelFromInstance(
    'PersistedModel', article, {idInjection: true}
  );

// Create an instance of 'Article' model with the contents of the
// 'article' document.
var modelInstance = new Article(article);
console.log(modelInstance.toObject());

// Persist the 'article' model instance in the data source.
Article.create(article, function (err, createdmodel) {
  console.log('Created: ', createdmodel.toObject());
  Article.findById(createdmodel.id, function (err, foundmodel) {
    console.log('Found: ', foundmodel.toObject());
  });
});

callback();
}); // ds.once
}); // introspection
```

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-31. Example: Create a model from introspection (2 of 2)

The `buildModelFromInstance` function creates a LoopBack model with the properties in the Article JSON document. This LoopBack model extends the `PersistedModel` base class. As a persisted model, you can create, retrieve, update, and delete model objects. Keep in mind that Article represents the data structure of the JSON document. It does not store any instance data.

The `new Article(article)` statement creates an instance of the Article model.

The `Article.create` function persists the Article model instance into the memory data source. After you create the model in the data source, the callback function retrieves the Article model instance from the data source.

Unit summary

- Explain the role of a LoopBack data source
- Identify the role of database and non-database connectors
- Install a LoopBack connector
- Explain how to map data sources to models
- Generate an application that uses the built-in memory connector
- Build model instances from unstructured data

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-32. Unit summary

Review questions

1. True or False: You must extend the persisted model base class to use a database connector.
2. Which LoopBack connector can generate models from a relational data source?
 - A. MySQL connector
 - B. Cloudant connector
 - C. MongoDB connector
 - D. REST connector



Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-33. Review questions

Write your answers here:

- 1.
- 2.

Review answers

1. True or False: You must extend the persisted model base class to use a database connector.

The answer is True.



2. Which LoopBack connector can generate models from a relational data source?

- A. MySQL connector
- B. Cloudant connector
- C. MongoDB connector
- D. REST connector

The answer is A.

Exercise: Define LoopBack data sources

Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-35. Exercise: Define LoopBack data sources

Exercise objectives

- Install and configure the MySQL connector
- Install and configure the MongoDB connector
- Generate models and properties from a data source
- Define relationships between models
- Test an API with the LoopBack API Explorer



Defining data sources with connectors

© Copyright IBM Corporation 2020

Figure 7-36. Exercise objectives

Unit 8. Implementing remote methods and event hooks

Estimated time

00:45

Overview

This unit explores how to define custom remote methods and hooks. You learn how to extend the data-centric LoopBack model with remote methods. You also learn how to implement event-driven functions with remote and operation hooks.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Explain the purpose of a remote method
- Define and call a remote method
- Explain the purpose and use cases for a remote hook
- Explain the purpose and use cases for an operation hook

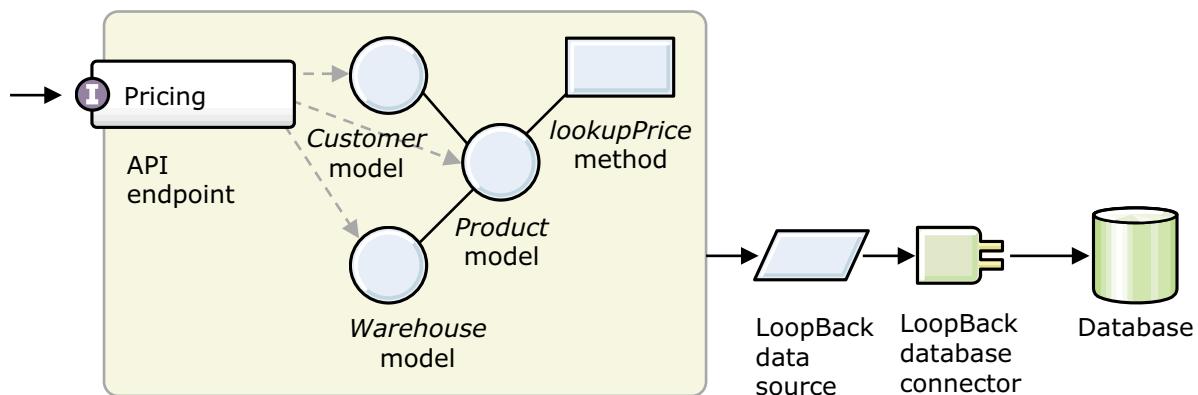
Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-1. Unit objectives

LoopBack framework: Remote methods and hooks

- The framework generates only a set of API operations that create, retrieve, update, and delete model and model properties
- To implement free-form API operations, create **remote methods** in the model
- To implement processing logic before and after API operations, create **remote hooks** in the model



Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-2. LoopBack framework: Remote methods and hooks

The *lookupPrice* function is an example of a remote method. It returns the price of a product based on business rules that you implement as a Node.js function. Since the *lookupPrice* function does not map to the database-centric create, retrieve, update, or delete operations, you must manually define this operation in the *Product* model.

Remote hooks are event handlers that run before and after an API operation. For example, you can define a remote hook that listens and logs each *lookupPrice* request from the pricing application. In the example that follows, you learn how to write a remote hook that saves the calculated price for a product after a client calls the *lookupPrice* API operation.

Steps: LoopBack remote methods and hooks

In this topic, you focus on the following LoopBack application implementation step:

1. Generate a LoopBack application
2. Configure a data source to persist LoopBack model data
3. Define models, properties, and relationships
4. **Add remote methods and hooks** to add custom business logic
5. Review and test the API with the API Explorer web application

Figure 8-3. Steps: LoopBack remote methods and hooks

In this topic, you focus on the following LoopBack application implementation step:

Add remote methods and hooks to add custom business logic.

Adding logic to models

You can add custom logic to a model in three places:

1. **Remote methods** map API operations to a Node function
2. **Remote hooks** define a method that LoopBack calls before or after it runs a remote method
3. **Operation hooks** define a method that LoopBack calls when an API consumer calls a create, retrieve, update, or delete operation on a model

Figure 8-4. Adding logic to models

You can add custom logic to a model in three places:

1. Remote methods map API operations to a Node function.
2. Remote hooks define a method that LoopBack calls before or after it runs a remote method.
3. Operation hooks define a method that LoopBack calls when an API consumer calls a create, retrieve, update, or delete operation on a model.

What is a remote method?

- A **remote method** is a static method that LoopBack exposes as an API operation
 - Implement a remote method to perform tasks that the LoopBack standard model API does not cover
- You add two components to the **model script** file to create a remote method:
 1. Implement a `static` method to handle the API operation request
 2. Call `remoteMethod()` to register the static method

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-5. What is a remote method?

A remote method is a static method that LoopBack exposes as an API operation. Implement a remote method to do tasks that the LoopBack standard model API does not cover.

The LoopBack standard model REST API includes the create, retrieve, update, and delete operations on model properties. For more information, see:

<https://loopback.io/doc/en/lb3/PersistedModel-REST-API.html>

You add two components to the model script file to create a remote method. First, implement a static method to handle the API operation request. Second, call `remoteMethod()` to register the static method.

Example: Remote method

```
module.exports = function(Product) {
  var priceService;
  Product.on('attached', function() {
    priceService = Product.app.dataSources.priceREST;
  });

  Product.lookupPrice = function() {
    priceService.price.apply(priceService, arguments);
  };

  Product.remoteMethod('lookupPrice', {
    description: 'Calculate price for specified product',
    accepts: [{ arg: 'id', type: 'number' }],
    returns: { arg: 'price', root: true },
    http: { verb: 'get' }
  });
};
```

common/models/**product.js**

[Implementing remote methods and event hooks](#)

© Copyright IBM Corporation 2020

Figure 8-6. Example: Remote method

In this example, the **product.on** event handler saves a reference to the priceREST data source in a local variable, `priceService`. The LoopBack framework initializes and loads its data sources asynchronously. Therefore, you must define an event handler to run after the framework initializes the REST data source.

The **lookupPrice** function contains the logic for the remote method. It calls a resource from the `priceService` REST data source.

The **remoteMethod** function call registers the API operation description, input, and output parameters for the **lookupPrice** remote method.

What is a remote hook?

- A remote hook is a function that LoopBack runs before or after it calls a remote method
 - Use **beforeRemote** hooks to validate inputs to a remote method
 - Use **afterRemote** hooks to modify, log, and access the response from a remote method before LoopBack sends the result to the API caller
- Three methods can be implemented for each model:
 - `beforeRemote()` runs before the remote method
 - `afterRemote()` runs after the remote method successfully completes
 - `afterRemoteError()` runs after the remote method completes with an error
- To retrieve information on the API call, the context `ctx` object contains the request, response, and access token of the user that calls the remote method

Figure 8-7. What is a remote hook?

A remote hook is a function that LoopBack runs before or after a remote method of model is called.

Three methods can be implemented for each model:

The `beforeRemote()` hook runs before the remote method.

The `afterRemote()` hook runs after the remote method successfully completes.

The `afterRemoteError()` hook runs after the remote method completes with an error.

Use **beforeRemote** hooks to validate inputs to a remote method.

Use **afterRemote** hooks to modify, log, and access the response from a remote method before LoopBack sends the result to the API caller.

Example: Remote hook

```
module.exports = function(Item) {
  Item.afterRemote('prototype.__create__reviews', {
    function (ctx, remoteMethodOutput, next) {
      var itemId = remoteMethodOutput.itemId;
      var searchQuery = {include: {relation: 'reviews'}};

      Item.findById(itemId, searchQuery,
        function findItemReviewRatings(err, findResult) {
          var reviewArray = findResult.reviews();
          var reviewCount = reviewArray.length;
          var ratingSum = 0;

          for (var i = 0; i < reviewCount; i++) {
            ratingSum += reviewArray[i].rating;
          }
          var updatedRating = ratingSum / reviewCount;
          console.log("new calculated rating: " + updatedRating);
          next();
        });
    });
};
```

common/models/item.js

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-8. Example: Remote hook

This remote hook example is taken from the exercise that accompanies this presentation unit. In this scenario, the user creates an item review with the POST /review API operation. After the create operation completes, LoopBack runs the **afterRemote** remote hook before it returns a response to the user.

The code within the remote hook searches for all reviews on the same item. Calculate the average review rating and display the updated rating in the console log.

What is an operation hook?

- An **operation hook** is a method that LoopBack runs before or after a high-level create, retrieve, update, or delete operation
- You can intercept actions that modify model data, independent of the specific method that calls them
 - For example, the **persists** operation hook intercepts the actions of the **create**, **upsert**, and **findOrCreate** operations from any model that extends the persisted model base class
- Developing an operation hook requires an understanding of the model API and the underlying LoopBack classes that implement the API
- For more information, see:
<https://loopback.io/doc/en/lb3/Operation-hooks.html>

Figure 8-9. What is an operation hook?

An operation hook is a method that LoopBack runs before or after a create, retrieve, update, or delete operation.

With operation hooks, you can intercept actions that modify model data.

Types of operation hooks

Type	Description	Use cases
access	Triggered when a LoopBack model queries a data source for data	Log access to model data by custom LoopBack functions, remote methods, API operations
before save / after save	Triggered before / after any operation creates or updates a PersistedModel object	Log changes to model record data
before delete / after delete	Triggered before / after any operation removes a PersistedModel object	Log record deletion
loaded	Triggered after LoopBack retrieves data from a data source, but before it creates a model object	<ul style="list-style-type: none"> Transform data from a data source before LoopBack creates a model <ul style="list-style-type: none"> For example, decrypt data for model properties
persist	Triggered before LoopBack persists model data to a data source	<ul style="list-style-type: none"> Transform data before storing in a data source <ul style="list-style-type: none"> For example, encrypt data from model

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-10. Types of operation hooks

When you define an operation hook, you create a function that listens to one of the seven operation events in the table.

The **access** hook fires whenever any PersistedModel object accesses a LoopBack data source. You can define access operation hooks to log and monitor data access through a LoopBack connector.

The **before save** and **after save** events fire before and after a function creates or updates data in a PersistedModel object. Conversely, the **before delete** and **after delete** events fire before and after a function deletes a PersistedModel object. With these two pairs of events, you can monitor and check on changes to persisted model objects.

The **loaded** event fires after the LoopBack framework retrieves raw data from a data source, but before it creates an instance of a model object with the data. The **persist** event fires before the LoopBack framework writes data to a data source. The purpose of the **loaded** and **persist** events is to transform the data before the framework persists the data and after it retrieves the data from a data source.

Which operations trigger operation hooks?

Method name	Hooks involved
all, find, findOne, findById, exists, count	access, loaded
create	before save, after save, loaded, persist
upsert (updateOrCreate)	access, before save, after save, loaded, persist
findOrCreate	access, before save, after save, loaded, persist
deleteAll (destroyAll), deleteById (destroyById)	access, before delete, after delete
updateAll	access, before save, after save, persist
prototype.save	before save, after save, persist, loaded
prototype.delete	before delete, after delete
prototype.updateAttributes	before save, after save, loaded, persist

Figure 8-11. Which operations trigger operation hooks?

When `findOrCreate` finds an existing model, the save hooks are not triggered. However, connectors that provide atomic implementation might trigger before save hook even when the model is not created, since they cannot determine in advance whether the model is created or not.

Example: Access operation hook

```
module.exports = function(Item) {
  Item.observe('access', function logQuery(ctx, next) {
    console.log(
      'Accessed %s matching %j',
      ctx.Model.modelName, ctx.query.where
    );
    next();
  });

  Item.observe('access', function limitAccess(ctx, next) {
    ctx.query.where.tenantId =
      loopback.getCurrentContext().tenantId;
    next();
  );
};

});
```

common/models/item.js

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-12. Example: Access operation hook

The **Item.observe** function registers a Node.js function with an operation hook event. In this example, the **logQuery** function records the model that the LoopBack framework queried, along with the query **where** clause. The **where** clause is a filter that limits the result set in an **Item.find** operation.

The second operation hook is an example of how the function can modify the search parameters before the LoopBack framework queries the data source. In this example, the **limitAccess** function modifies the **tenantId** property so that the current application can query models that belong to the tenant only. The **loopback.getCurrentContext()** function retrieves data from the current LoopBack application. This function is not related to the **ctx** variable, which represents the context in which the LoopBack framework accessed the model object.

At the end of every operation hook function, you must call the **next()** function. This function passes control back to the LoopBack application, and signals the end of the operation hook function.

Example: After save operation hook

```
module.exports = function(Item) {
  Item.observe('after save', function logCreateUpdate(ctx, next) {
    if (ctx.instance) {
      console.log(
        'Saved %s#%s',
        ctx.Model.modelName, ctx.instance.id
      );
    } else {
      console.log(
        'Updated %s matching %j',
        ctx.Model.plural modelName, ctx.where
      );
    }
    next();
  });
};
```

common/models/item.js

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-13. Example: After save operation hook

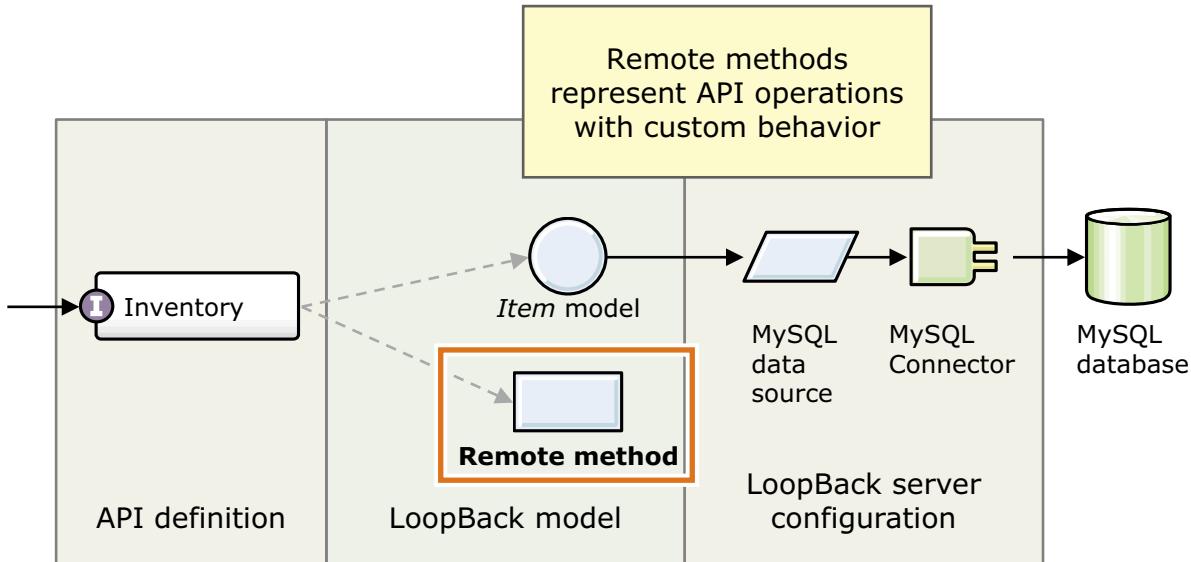
In this example, the `Item.observe` function creates an operation hook for the **after save** event. The `logCreateUpdate` function checks whether the LoopBack framework created or updated the **item** model object. If the framework created an instance of **item**, the `ctx.instance` property is a reference to the new instance. The function logs the identifier for the item model.

If the LoopBack framework updated an existing object, the `ctx.instance` property is undefined. In this second case, the function records the item model or models that the framework updated.

In the last step, the operation hook function calls the `next()` function. This function returns control back to the LoopBack framework.

When do you implement remote methods?

- Implement a **remote method** to model custom behavior that is not covered by the data-centric model operations



Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-14. When do you implement remote methods?

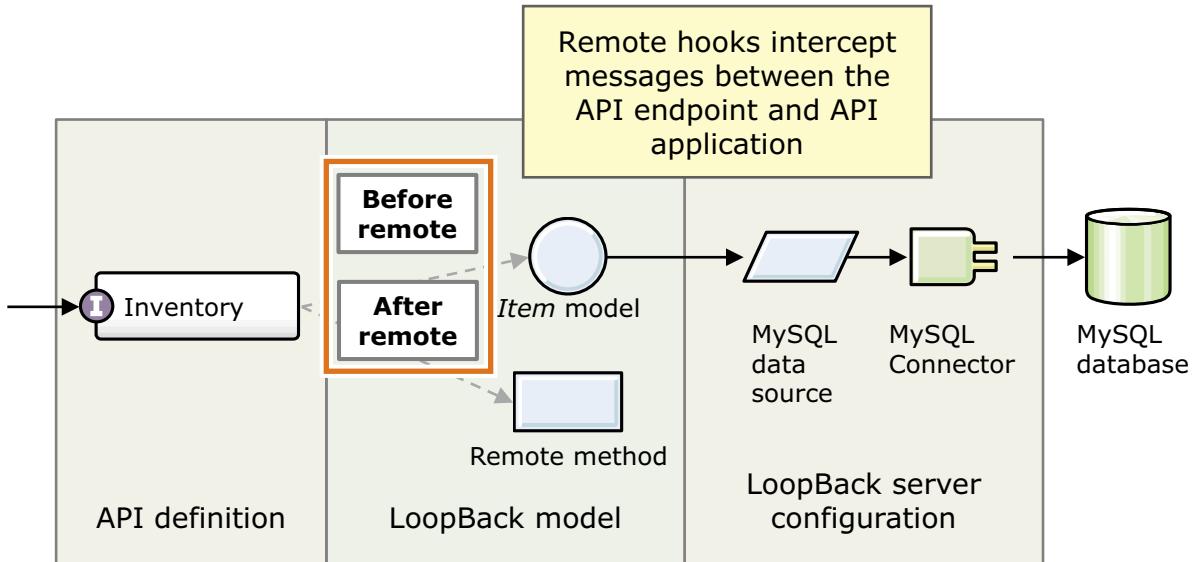
The LoopBack API application that you create has three logical components:

1. The OpenAPI definition specifies the interface for API operations, request, and response messages. At run time, the API gateway enforces the operations and policies in the API definition.
2. The LoopBack model consists of the model definition files and script files. Each model is a Node.js object that represents your business data.
3. The LoopBack server configuration is a set of JSON configuration files and Node.js packages that connect your model code to external systems. In the exercise case study, you created a MySQL data source to retrieve and persist data from a MySQL database.

You configure, install, or generate the API definition and server configuration. You implement custom code in the model script file, including remote methods. **Remote methods** are operations that you expose in your API definition. Unlike model API operations, remote methods do not correspond to the data-centric create, retrieve, update, and delete operations. They are free-form methods that you implement to cover custom behavior.

When do you implement remote hooks?

- Add **remote hooks** when you want to intercept, log, and modify information before or after a specific API operation



Implementing remote methods and event hooks

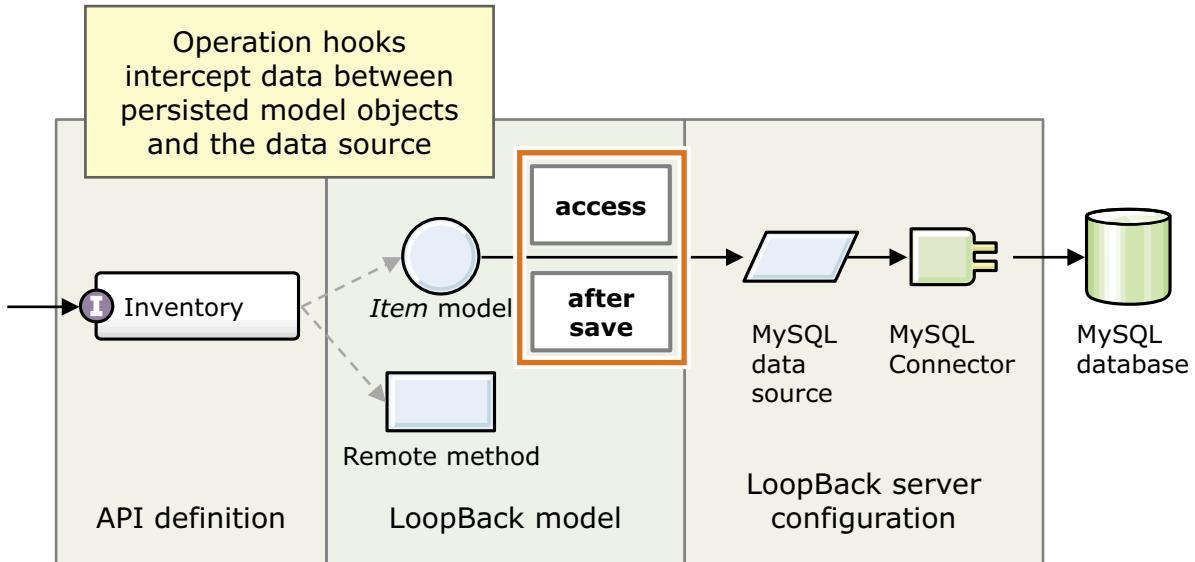
© Copyright IBM Corporation 2020

Figure 8-15. When do you implement remote hooks?

Remote hooks are another category of custom code that you implement in the model script file. These functions intercept requests to a specific API operation. You can trigger a function to log, transform, or modify the HTTP request and response message before or after an API operation.

When do you implement operation hooks?

- Add **operation hooks** to intercept data that flows between a model object and the LoopBack data source that persists and retrieve its data



Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-16. When do you implement operation hooks?

Operation hooks monitor activity between a persisted model object in your API application, and the data source that persists the model's data. You can define an operation hook that intercepts, logs, and transforms data as the LoopBack framework persists or retrieves model data. The functions that you define in an operation hook examine the model object through the `ctx` context variable.

Although remote hooks and operation hooks are both event-driven listener functions, they act in different layers of your LoopBack framework. Remote hooks act on API operations, while operation hooks act on a **class** of actions that your model performs on the data source. Operation hooks do not deal with the HTTP request and response message from the API operation call. It handles the data that the framework stores or retrieves from the data source.

Unit summary

- Explain the purpose of a remote method
- Define and call a remote method
- Explain the purpose and use cases for a remote hook
- Explain the purpose and use cases for an operation hook

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-17. Unit summary

Review questions

1. What is the difference between a remote hook and an operation hook?
 - A. The application calls an operation hook on data-centric operations on the model
 - B. The application triggers a remote hook before or after a remote method call
 - C. Operation hooks can apply to more than one model object
 - D. All of the above
2. How do you implement a free-form API operation in LoopBack?
 - A. You implement a Node.js function in the model script file
 - B. You implement a Node.js function in the model definition file
 - C. You declare a remote method in the model definition file
 - D. You define a path in the API definition file



Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-18. Review questions

Write your answers here:

- 1.
- 2.

Review answers

1. What is the difference between a remote hook and an operation hook?
 - A. The application calls an operation hook on data-centric operations on the model
 - B. The application triggers a remote hook before or after a remote method call
 - C. Operation hooks can apply to more than one model object
 - D. All of the above

The answer is D
2. How do you implement a free-form API operation in LoopBack?
 - A. You implement a Node.js function in the model script file
 - B. You implement a Node.js function in the model definition file
 - C. You declare a remote method in the model script file
 - D. You define a path in the API definition file

The answer is A



Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-19. Review answers

Exercise: Implement event-driven functions with remote and operation hooks

Implementing remote methods and event hooks

© Copyright IBM Corporation 2020

Figure 8-20. Exercise: Implement event-driven functions with remote and operation hooks

Exercise objectives

- Define an operation hook in a LoopBack application
- Define a remote hook in a LoopBack application
- Test remote and operation hooks in the LoopBack Explorer



Unit 9. Assembling message processing policies

Estimated time

01:00

Overview

In the API Gateway, message processing policies log, route, and transform API request and response messages. This unit explores the concept of message processing policies. You learn how to define a set of message processing policies in your API definition file with the API Manager.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Explain the concept of non-functional requirements
- Identify use cases for message processing policies
- Explain the relationship between message processing policies and the API application
- Identify the policies that the DataPower API gateway type supports

What is a message processing policy?

- A **message processing policy** is an action that **transforms**, **validates**, or **routes** API request and response messages at run time
- The documentation and toolkit also refer to message processing policies as gateway policies, or policies
- Policies do not implement API operations
 - Message processing policies maintain and enforce the non-functional requirements of an API
 - The API implementation fulfills the functional requirements of an API
- Message processing policies describe how the API gateway maintains the quality of service, rather than the operation behavior

Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-2. What is a message processing policy?

A message processing policy is an action that transforms, validates, or routes API operations at the HTTP request and response message level. At run time, the API gateway enforces the policies that you define as part of an API definition.

The API Manager refers to these configured actions as **policies**. To differentiate this type of policy against other policies, this course uses the term **message processing policy**.

Although you can build conditional logic constructs and API calling actions with policies, the purpose of policies is not to implement API operations. Message processing policies maintain and enforce the non-functional requirements of an API at the API gateway. You implement the operations of an API with an application that is hosted elsewhere in your architecture.

Message processing policies at run time

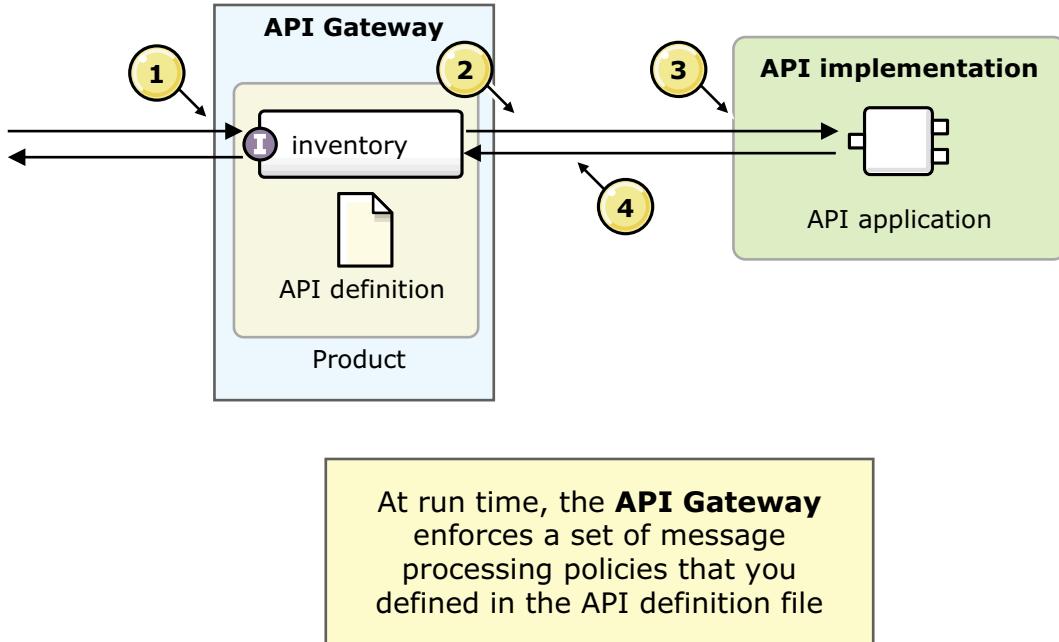


Figure 9-3. Message processing policies at run time

You publish an API definition, product, and plan to the API Manager. API Manager sends the message processing policy to the API gateway and configures an API endpoint according to the API definition.

1. When the API gateway receives an HTTP request, it validates the message against the API definition.
2. The API gateway also enforces a series of message processing policies that are defined in the API definition. In the simplest case, the assembly includes one policy: to call the API implementation.
3. The API provider implementation receives the request. After it processes the request, it sends an HTTP response back to the API gateway.
4. If the assembly includes message processing policies after the **invoke** policy, the gateway runs these policies before the HTTP response message is sent back to the client application.



Assembly editor: Creating policy assemblies

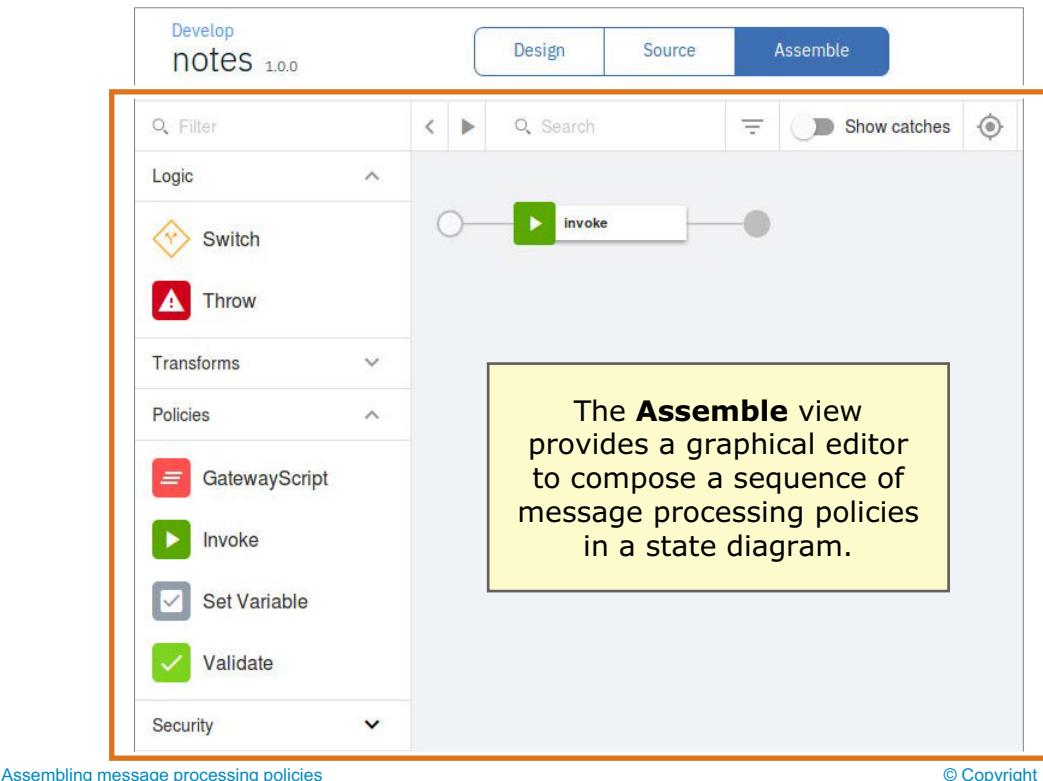


Figure 9-4. Assembly editor: Creating policy assemblies

The **assemble** view in the API Manager web application is a graphical editor for a sequence of message processing policies. The main parts consist of the **canvas**, **palette**, and various search bars.

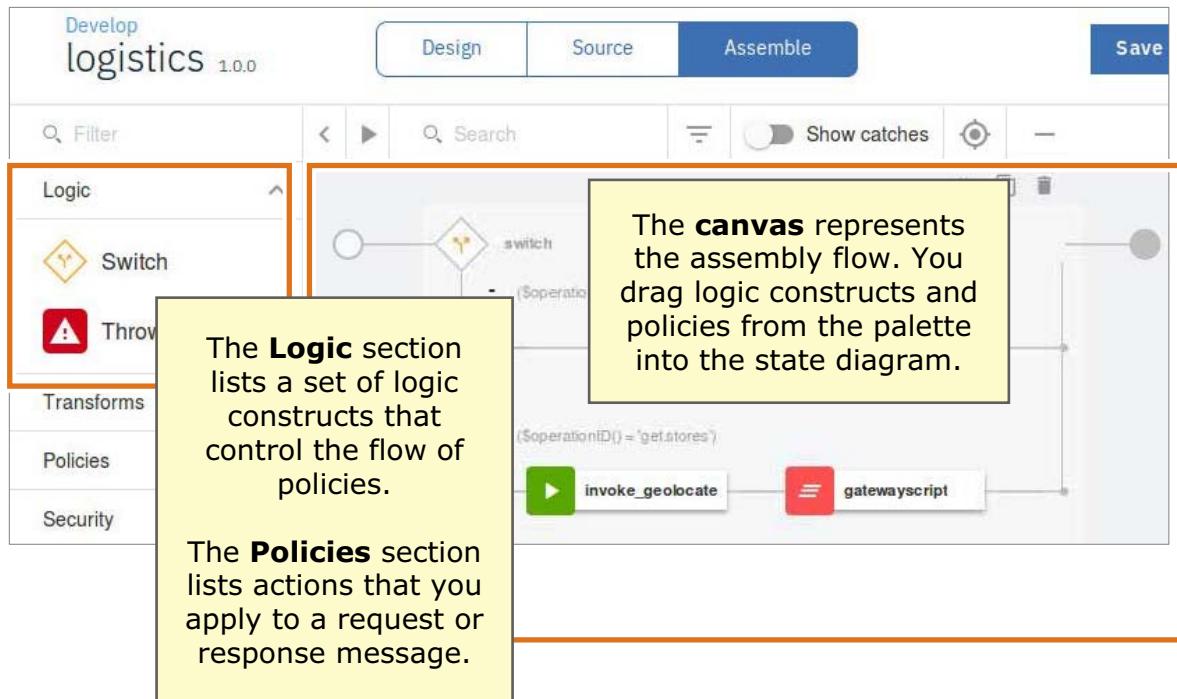
The API Manager application writes the policies and logic constructs in the **assemble** section of the **x-ibm-configuration** extension entry of the OpenAPI document when you save the API definition file.

The palette includes a number of expandable and collapsible drawers with the labels Logic, Transforms, Policies, and Security. When expanded, each drawer lists the available components.

The content of the palette depends on the gateway type that is selected for the API in the OpenAPI definition.



Assembly editor: Palette and canvas



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-5. Assembly editor: Palette and canvas

The **palette** lists the configuration constructions that you can place onto the **canvas**.

Logic constructs control the message flow across policies. For example, you can set a **switch** with a subflow for each API operation. Policies represent actions that the API Gateway runs on a request or response message.

The **canvas** represents the assembly flow: a sequence of policy actions that the API Gateway applies to HTTP request and response messages.

The open circle represents the incoming API request, and the filled circle represents the response that you return to the caller. You must have an **invoke** action that calls the API from the gateway. The policies to the left of the invoke policy apply to HTTP request messages. The policies to the right of the invoke policy apply to the HTTP response message.

Assembly editor: Filter, search, and gateway type



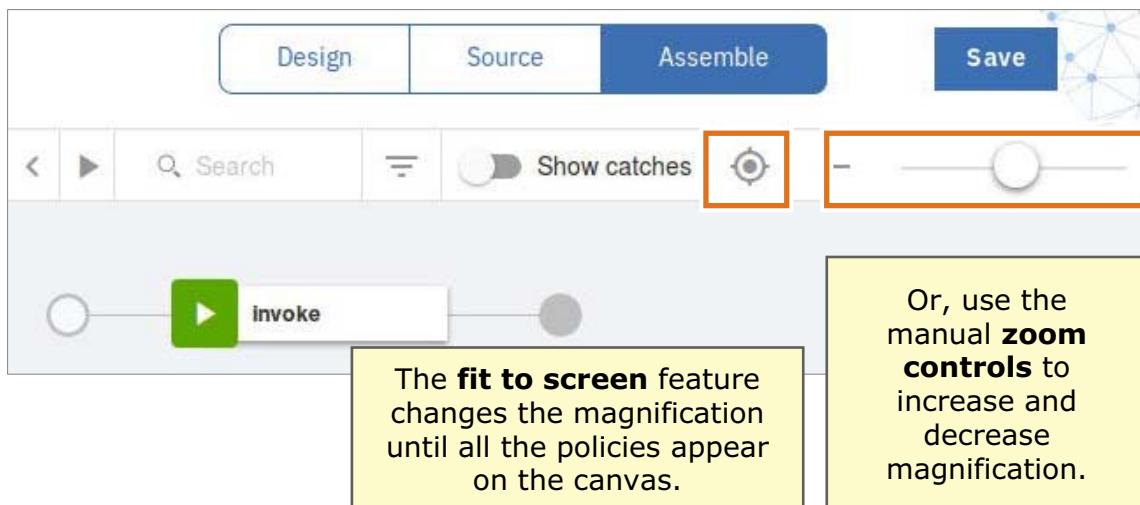
Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-6. Assembly editor: Filter, search, and gateway type

The quickest way to find a message processing policy is to enter the policy name into the **filter** bar. To find a policy or logic construct on the canvas, type the name in the **search** bar.

Assembly editor: Magnify and zoom



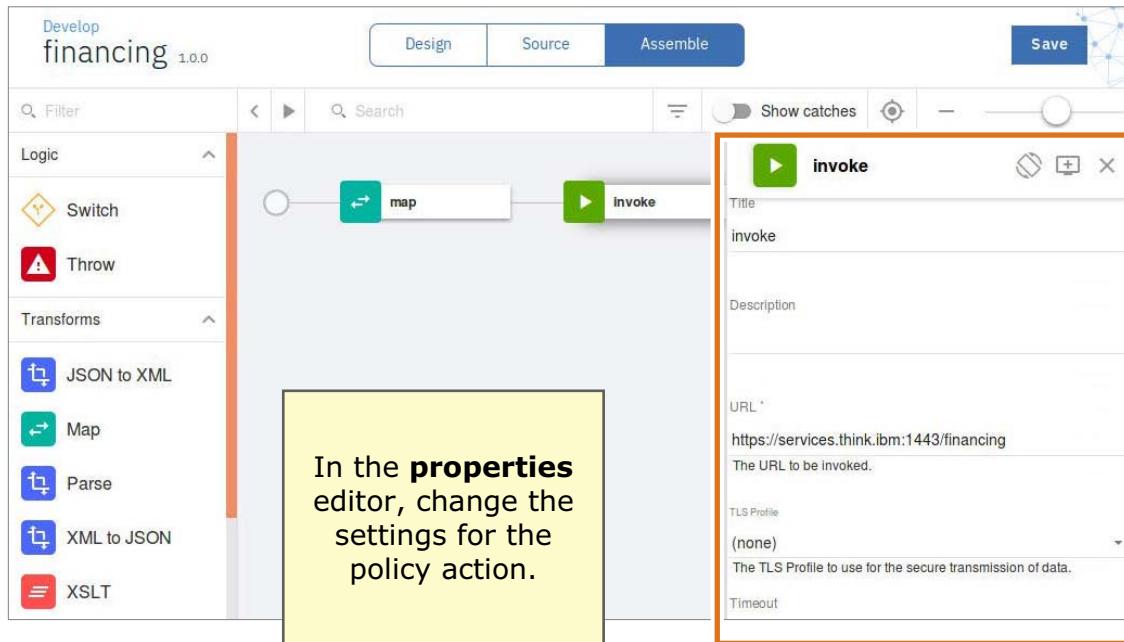
Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-7. Assembly editor: Magnify and zoom

The compass-like icon represents the **fit to screen** feature. When you select this option, the canvas zooms in or out until the entire row of message processing policies appear in view. You can also use the manual **zoom** controls to zoom in and out of a specific part of the canvas.

Assembly editor: Properties editor



Assembling message processing policies

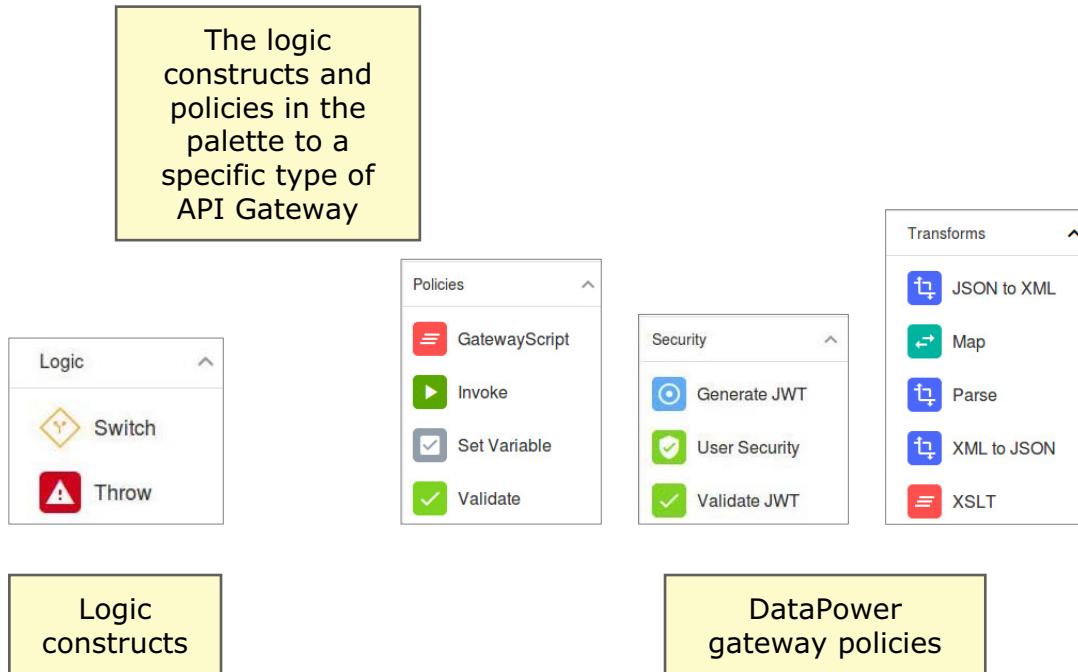
© Copyright IBM Corporation 2020

Figure 9-8. Assembly editor: Properties editor

The **properties** editor reveals more settings for the selected policy. In this example, the **invoke** policy is selected. The invoke properties view has a title, a description, a URL, TLS profile, timeout, and more settings. In this example, the URL is set to an actual endpoint address to which the gateway forwards the request.



API policies and logic constructs



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-9. API policies and logic constructs

The **palette** in the assembly editor divides the list of items into categories: logic constructs, policies, security, and transforms.

Logic constructs change the sequence in which the gateway runs policy actions.

Policies, Security, and Transforms perform an action on the HTTP message, or an environment variable.

You must specify which type of gateway each API uses in the OpenAPI API definition. The two gateway types are the DataPower API gateway and the DataPower gateway (v5 compatible).

Each DataPower gateway supports its own set of logic, policies, transforms, and security policies.

For example, an **operation-switch** component is supported by the DataPower gateway (v5 compatible) gateway type. For the DataPower API gateway, the same function is provided by the **switch** component. In some cases, the same policy is supported by both gateway types, but with a different version number.

These policies are not interchangeable – you must use the correct policy according to the gateway type.

The palette shows the available policies according to the gateway type that is specified for the API in the OpenAPI definition.

When you modify your API definitions to use a specific gateway type, you must ensure that each policy and policy version in the API are supported by the gateway type.

Example scenarios for policy assemblies

1. Forward requests to an API implementation
2. Select a sequence of policies based on the API operation
3. Map responses from multiple API calls into a single response
4. Transform REST API requests into a SOAP service request
5. Validate properties in an HTTP request message
6. Store the request message payload into API analytics

Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-10. Example scenarios for policy assemblies

This list introduces separate example scenarios that you can author with API policies and logic constructs. This list is not exhaustive: policies cover many other message processing scenarios.

- In the simplest scenario, you proxy all API operations to an existing API implementation.
- You use an operation-switch construct to select a sequence of policies based on the API operation. This policy is a case statement that handles different API operations.
- Map responses from multiple API calls into a single response.
- Transform REST API requests to a SOAP service request.
- Validate properties in an HTTP request message.
- Store the request message payload into API analytics.

The following set of slides explains how to handle these scenarios with policies.

Example one: Forward an API call with the invoke policy



- When you create an API definition, API Manager forwards API operations with an **invoke** policy.
- The policy calls the API application with the following target URL:
 - `$(runtime-url)` : Name of server that hosts the API implementation.
 - `$(request.path)` : The path portion of an API operation.
 - `$(request.search)` : The HTTP query string with the question mark (?) delimiter.

Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-11. Example one: Forward an API call with the invoke policy

When you create an API definition, API Manager forwards API operations with an **invoke** policy.

The URL in the properties view is where the target endpoint URL is specified.

The URL can be an actual web endpoint address, for example,
<http://soapsample.mybluemix.net/ItemService>.

The URL in the invoke properties view can also be specified as an API Manager context variable, for example, `$(target_url)`.

In an assembly policy field that supports variable references, such as the properties view, use the syntax `$(variable)`.

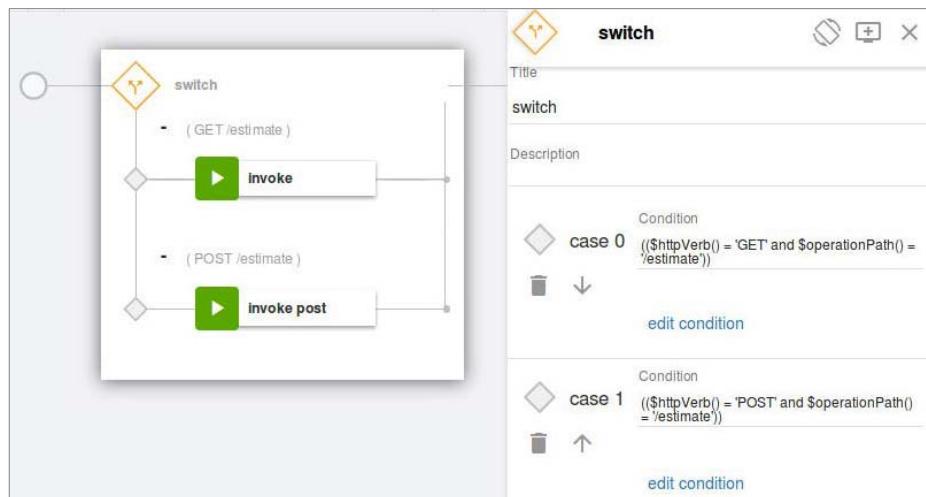
Variables can also be concatenated, for example:

`$(runtime-url)$(request.path)$(request.search)`.

This example is covered in exercise 2: Create an API definition from an existing API.

Example two: Switch case by API operation

- When the **switch** policy receives an API request, it selects a case that matches the operation condition.
 - Each case contains a sequence of API policies.
 - If none of the cases match the current API operation, the gateway runs the next policy after the operation-switch.



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-12. Example two: Switch case by API operation

The switch policy is used to select a case that matches the operation condition when an API request is received on the API Gateway type.

This scenario is covered in the second part of exercise 2: Create an API definition from an existing API.

Example three: Map multiple API calls into a response



- The **invoke** policy makes an HTTP request to any network endpoint.
 - You define the target URL and HTTP method for the service call
- You can define several **invoke** policies in a single assembly flow.
 - By default, the return message from the **invoke** policy overwrites the **response** message for the assembly flow
 - To avoid this behavior, save the response from each **invoke** policy into a different context variable
- Use a **map** policy to combine properties from several **invoke** policies into one API response message
 - In this example, the **map** policy combines the results from the **invoke_xyz** and **invoke_cek** policies

[Assembling message processing policies](#)

© Copyright IBM Corporation 2020

Figure 9-13. Example three: Map multiple API calls into a response

In this example, you make two consecutive calls to back-end services and then aggregate the responses into a single response with the map policy.

You can define several invoke policies in a single assembly flow.

By default, the return message from the invoke policy overwrites the response message for the assembly flow.

To avoid this behavior, save the response from each invoke policy into a different context variable.

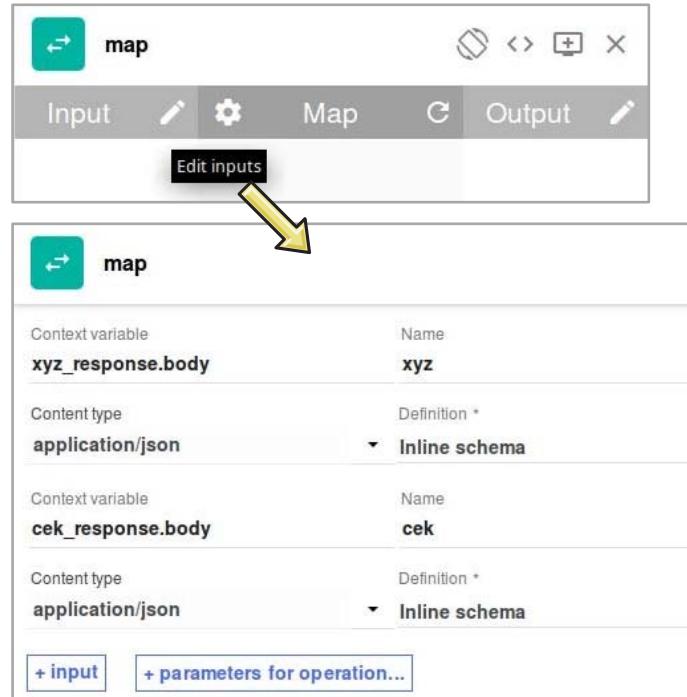
For example, in the properties dialog for invoke_xyz policy, define a context variable that is named xyz_response in the Response object variable field.

Then, in the properties dialog for invoke_cek policy, define a context variable that is named cek_response in the Response object variable field.

On the next page, you see how these context variables are used by the map policy of the assembly.

Example: Define input parameters in a map policy

- In the previous example, the **invoke** policies save the responses from remote API calls into two context variables:
 - **xyz_response**
 - **cek_response**
- The **input** column of the **map** policy reads the body of the response messages
- The **inline schema** makes sure that the responses from the remote API calls match the structure that you expect



Context variable	Name
xyz_response.body	xyz

Content type	Definition *
application/json	Inline schema

Context variable	Name
cek_response.body	cek

Content type	Definition *
application/json	Inline schema

+ input **+ parameters for operation...**

Figure 9-14. Example: Define input parameters in a map policy

In the previous example, the invoke policies save the responses from remote API calls into two context variables:

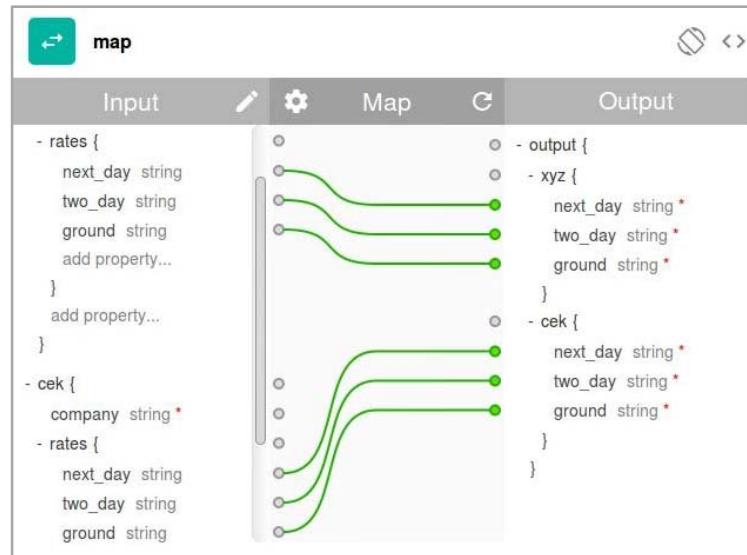
- **xyz_response**
- **cek_response**

The input column of the map policy reads the body of the response messages that are saved in the two response object variables.

The inline schema makes sure that the responses from the remote API calls match the structure that you expect.

Example: Map API results into a single response message

- In this example, you aggregate shipping rates from two delivery companies into the API response message
- In the **map** column, draw a **wire** between the nodes in the input and output columns to **copy** properties into the output message
- In the **output** column, you define the structure of the message payload after the **map** policy



[Assembling message processing policies](#)

© Copyright IBM Corporation 2020

Figure 9-15. Example: Map API results into a single response message

In this example, you aggregate shipping rates from two delivery companies, named xyz and cek, into the API response message.

In the map column, draw a wire between the nodes in the input and output columns to copy properties into the output message.

In the output column, you define the structure of the message payload after the map policy.

This example is covered in the optional exercise 13: Assemble the shipping message processing policy.

Example four: Transform REST to SOAP



- In this example, an REST API operation calls a SOAP service, and returns the result as a JSON object in an HTTP message.
- The **map** policy copies input parameters from an REST API request and saves it in an XML SOAP message payload.
- The **invoke** policy sends the SOAP message to a remote service.
- The **gatewayscript** policy sets the response message content-type to application/xml
 - Some SOAP services set the content-type to application/soap+xml instead of application/xml
- The **parse** policy validates the message contents
- The **xml-to-json** policy converts the SOAP response message into a JSON object.

[Assembling message processing policies](#)

© Copyright IBM Corporation 2020

Figure 9-16. Example four: Transform REST to SOAP

In this example, an REST API operation calls a SOAP service, and returns the result as a JSON object in an HTTP message.

This scenario is covered in the financing API of exercise 7: Assemble message processing policies.

Example: Define input parameters in a map policy

- In the **transform REST to SOAP** example, the REST API operation has three input parameters:
 - amount
 - duration
 - rate
- Copy the three request parameters into the **input** column of the **map** policy
 - The **request** context variable represents the initial request to the current API operation

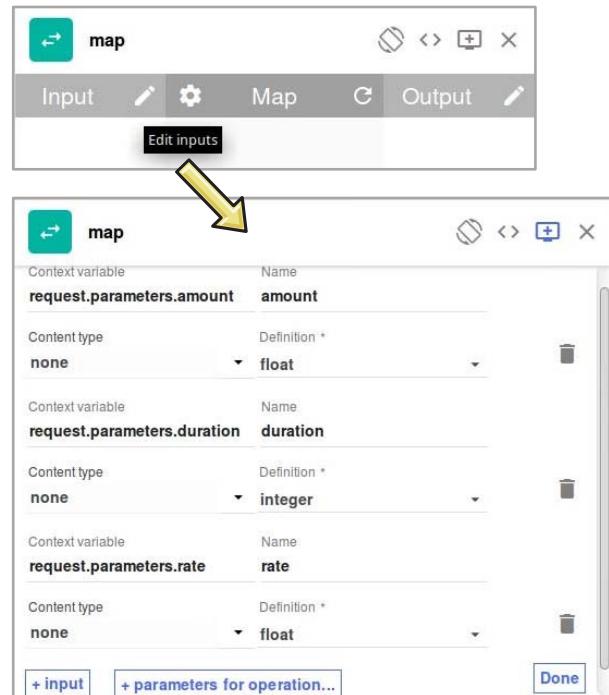


Figure 9-17. Example: Define input parameters in a map policy

In the transform REST to SOAP example, the REST API operation has three input parameters:

- amount
- duration
- rate

Copy the three request parameters into the input column of the map policy.

The request context variable represents the initial request to the current API operation.

Example: Map REST parameters to a SOAP request

- Define a SOAP envelope in the **output** column in the **map** policy.
 - The SOAP envelope is an XML message that contains the input parameters for a SOAP service operation.
- In the following step in the assembly, add an **invoke** policy to send the SOAP request message to the SOAP service.

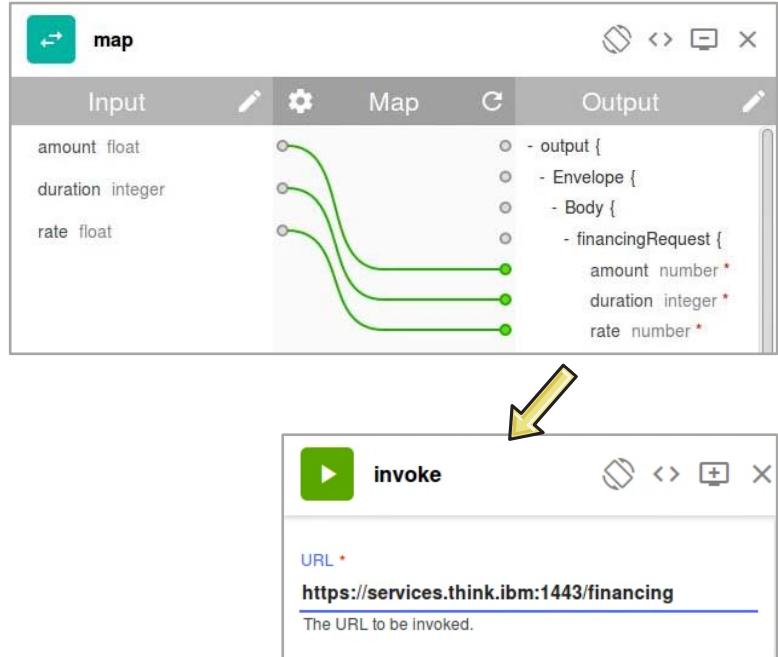


Figure 9-18. Example: Map REST parameters to a SOAP request

In the transform REST to SOAP example, this step maps REST parameters to a SOAP request.

Define a SOAP envelope in the output column in the map policy based on an inline schema.

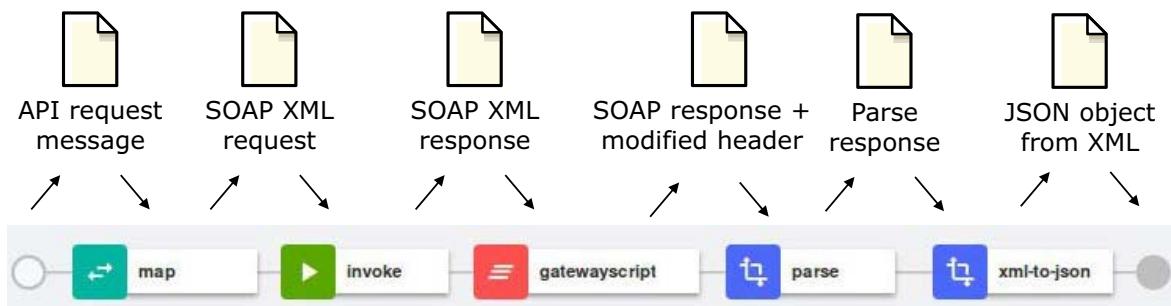
The SOAP envelope is an XML message that contains the input parameters for a SOAP service operation.

Map the REST parameters by connecting the REST input parameter fields to the corresponding like-named SOAP fields.

In the following step in the assembly, add an invoke policy to send the SOAP request message to the SOAP service.

What is the message payload?

- The message **payload** is the current message state in the assembly flow.
 - At the start of the assembly flow, the original **request message** is the payload.
 - You can modify, transform, or even replace the **payload** with API policies in the assembly flow.
 - At the end of the assembly flow, the gateway sends the **payload** as the **response message** to the API call.



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-19. What is the message payload?

In this example, an REST API operation calls a SOAP service, and returns the result as a JSON object in an HTTP message.

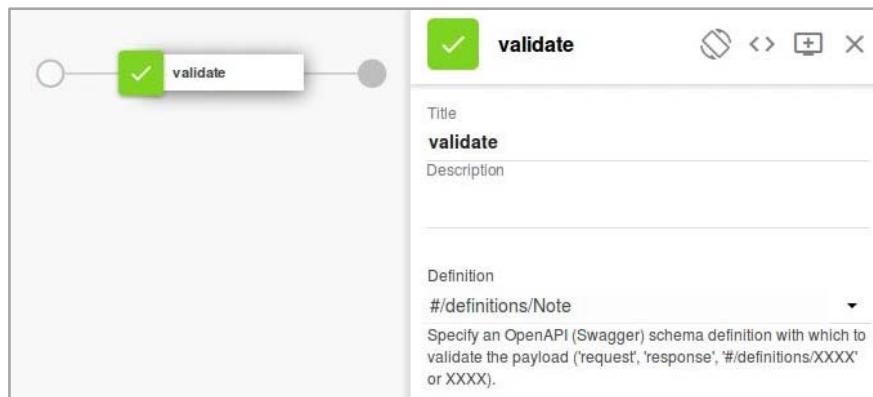
The message **payload** is the current message state in the assembly flow.

At the start of the assembly flow, the original **request message** is the payload.

You can modify, transform, or even replace the **payload** with API policies in the assembly flow.

At the end of the assembly flow, the gateway sends the **payload** as the **response message** to the API call.

Example five: Validate properties in an HTTP message



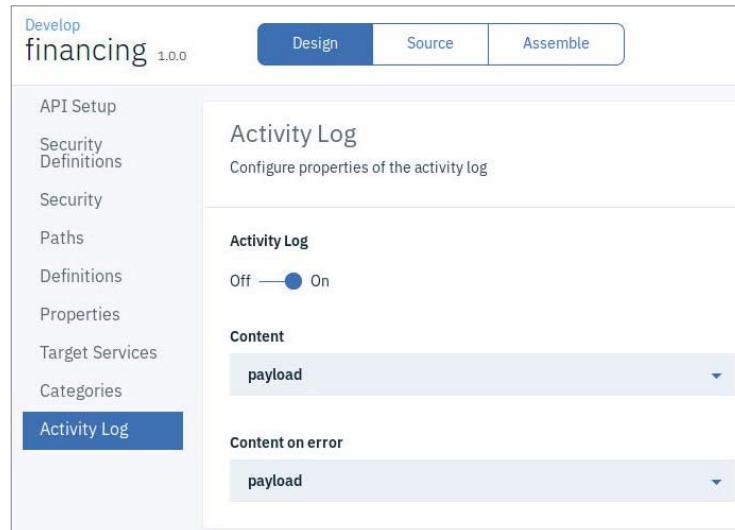
- Use the **validate** policy to check whether the message **payload** matches a defined schema type.
 - To validate the original request message, add **validate** with the **request** definition at the start of the flow
 - To validate an intermediate response, add **validate** with a **custom definition**.
 - To validate the API response message, add **validate** with the **response** definition after the policies that collate the final response message.

Figure 9-20. Example five: Validate properties in an HTTP message

Use the validate policy to check whether the message payload matches a defined schema type at any stage in the message processing sequence.

Example six: Store message payload in API analytics

- For a DataPower API gateway type, use the **Activity Log** tab in API Manager **Design view** to configure logging preference for the API activity that is stored in Analytics



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-21. Example six: Store message payload in API analytics

For a **DataPower API** gateway, use the **Activity Log** tab in API Manager **Design view** to configure logging preference for the API activity that is stored in Analytics.

The information that the policy collects is saved in an API event record, a log entry that captures the metadata in each API execution event.

Choose whether to save metadata on the invocation URL (activity), the activity and header (header), or the activity and message payload (payload).

You can set different settings for normal operation (content) or when an error occurs when the API is called (content on error).

You must associate the API Connect analytics feature with the selected gateway type in Cloud Manager to retrieve the activity log.

The activity-log policy captures four types of content:

- The **none** setting indicates that no logging occurs.
- The **activity** setting stores the resource URL that the client called. This option is the default setting for normal operation.
- The **header** setting stores the activity and the request header.

- The **payload** setting stores the activity, the request header, and the request message body. This option is the default setting for an error event.

Note: If you set the activity-log level to **none**, the option disables notifications for application developers who use your Developer Portal.

**Note**

If the API is defined to use the DataPower API Gateway, then the Activity Log tab in API Manager replaces the **activity-log** policy that is defined in the assembly flow for the DataPower Gateway (v5 compatible) type.

Unit summary

- Explain the concept of non-functional requirements
- Identify use cases for message processing policies
- Explain the relationship between message processing policies and the API application
- Identify the policies that the DataPower API gateway type supports

Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-22. Unit summary

Review questions

1. True or False: It is not a recommended practice to implement API operations with policies.
2. What is the message payload?
 - A. The payload is the input parameters of an API operation.
 - B. The payload is the API request header.
 - C. The payload is the API parameter list.
 - D. The payload is a buffer that the policy assembly uses to process or construct an API response message.
3. Which policy is only available on the DataPower API Gateway type?
 - A. Validate.
 - B. Switch.
 - C. Invoke.
 - D. Set Variable.



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-23. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: It is not a recommended practice to implement API operations with policies.
The answer is True.

2. What is the message payload?
 - A. The payload is the input parameters of an API operation.
 - B. The payload is the API request header.
 - C. The payload is the API parameter list.
 - D. The payload is a buffer that the policy assembly uses to process or construct an API response message.
The answer is D.

3. Which policy is only available on the DataPower API Gateway type? The answer is B.
 - A. Validate.
 - B. Switch.
 - C. Invoke.
 - D. Set Variable.



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-24. Review answers

Exercise: Assemble message processing policies

Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-25. Exercise: Assemble message processing policies

Exercise objectives

- Import an OpenAPI definition into API Manager
- Test the OpenAPI definition in API Manager
- Create an API with JSON object definitions and paths
- Configure an API to call an existing SOAP service
- Import an existing API definition into the source editor
- Create an assembly with a switch that has a flow for each API operation
- Define a gateway script with an API assembly that saves a variable and calls an external map service
- Test DataPower gateway policies in the API Manager assembly view



Assembling message processing policies

© Copyright IBM Corporation 2020

Figure 9-26. Exercise objectives

Unit 10. Declaring client authorization requirements

Estimated time

01:00

Overview

This unit explores how to define client authorization requirements in the API definition. The client authorization requirements specify which authentication and authorization standards to enforce. You learn how to configure API keys, HTTP basic authentication, and OAuth 2.0 authorization schemes.

How you will check your progress

- Review questions

Unit objectives

- Identify the security definition options in API Connect
- Explain the concept and use cases for API keys
- Explain the concept and use cases for HTTP basic authentication
- Explain the concept and use cases for OAuth 2.0 authorization
- Explain the steps in the OAuth 2.0 message flow

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-1. Unit objectives

10.1. API security concepts

API security concepts

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-2. API security concepts

Topics

API security concepts

- Identify client applications with API key
- Authenticate clients with HTTP basic authentication
- Introduction to OAuth 2.0

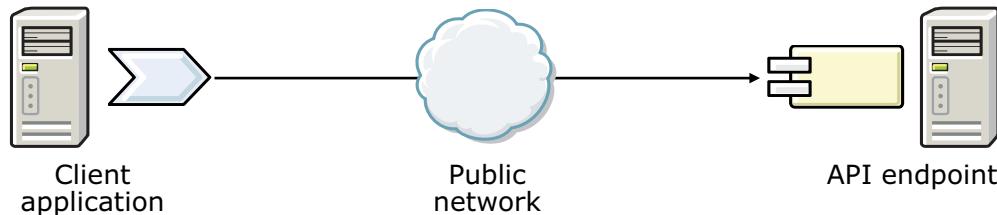
Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-3. Topics

Authentication and authorization: API security definitions

- To enforce authentication and authorization for your API, define and apply security definitions in your API definition
 - Your gateway **authenticates** users to verify the identity of the client
 - The gateway **authorizes** access to an API operation for clients that you allow
- API security definitions do not handle all aspects of API security
 - For example, you define transport level security (TLS) providers in the IBM API Management Server
- Not every API needs to be secured
 - Some resources might not contain sensitive information



[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-4. Authentication and authorization: API security definitions

To enforce authentication and authorization for your API, define and apply security definitions in your API definition. Your gateway authenticates users to verify the identity of the client. The gateway authorizes access to an API operation for clients that you allow.

This unit discusses how to authenticate and authorize API clients with IBM API Connect. You must consider other aspects of API security, but API security definitions do not cover those aspects. For example, API security definitions do not cover encryption and integrity. In API Manager, you define transport level security (TLS) profiles to specify the keys and certificates that secure data transmission over a network.

Last, consider the fact that not every API needs to be secured. Some resources might not contain sensitive information.

How do you secure your APIs in API Connect?

1. Create a **security definition**

- The security definition states which security scheme API Connect applies to your API
- The definition specifies the configuration settings for the scheme

2. Enable a security definition to your API

- To call an API operation, the client application must provide the information that you specified in the security definition
- You can apply a security definition to an entire API, or a specific operation within an API

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-5. How do you secure your APIs in API Connect?

The security definitions that you create in an API definition configure the client authentication and authorization schemes.

What types of security definitions can you define?

Definition type	Description
API key	The API key scheme authenticates the API caller from the client ID and client secret credentials
Basic	The HTTP basic authentication scheme enforces authentication and authorization at the HTTP message protocol layer
OAuth 2.0	The OAuth 2.0 scheme is a token-based authentication protocol that allows third-party websites to access user data without requiring the user to share personal information

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-6. What types of security definitions can you define?

What types of security definitions can you define?

In API Connect, the three security definitions configure client authentication and authorization for API clients.

The API key scheme authenticates the API caller from the client ID and client secret credentials. The HTTP basic authentication scheme enforces authentication and authorization at the HTTP message protocol layer. The OAuth 2.0 scheme is a token-based authentication protocol that third-party websites can use to access user data without requiring the user to share personal information.

Other security aspects, such as message encryption, are not covered in the security settings in your API definition.

10.2. Identify client applications with API key

Identify client applications with API key

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-7. Identify client applications with API key

Topics

- API security concepts
- Identify client applications with API key
- Authenticate clients with HTTP basic authentication
- Introduction to OAuth 2.0

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-8. Topics

API key: A unique client application identifier

The screenshot shows two interface panels. The top panel is titled 'Security Definitions' under 'API Setup'. It lists a single entry: 'clientIdHeader' (NAME), 'apiKey' (TYPE), and 'header' (LOCATED_IN). The bottom panel is titled 'API Security Definition' and contains fields for 'Name' (set to 'clientIdHeader'), 'Description (optional)', 'Type' (radio button selected for 'API Key', with 'Basic' and 'OAuth2' options available), and 'Located In' (set to 'Header'). An orange box highlights the 'clientIdHeader' entry in the security definitions list, and another orange box highlights the 'API Key' selection in the type dropdown.

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-9. API key: A unique client application identifier

The API key uniquely identifies the client application. If you enable **clientID** as a security requirement in your API, the client must provide its client ID on every API operation call. In addition to establishing the client identity, API Connect uses the client ID value for analytics and to enforce operation quotas.

The client secret is a unique value that API Connect generates. When a client developer registers an application, the API Connect Developer Portal creates and provides the client ID and client secret.

You can create a new OpenAPI in API Manager and *clear* the option to **Secure using Client ID** during creation. A security definition that is named **clientIdHeader** with an API Key is still added to the security definitions for the API. The clientIdHeader security definition defaults to an API Key type that specifies the Client ID as the unique identifier.

If you want the API to use the API Key, you must still add it in the Security tab of the Design view for the API.

- The **API key** scheme defines two types of security metadata:
 - The **Client ID** is a unique identifier for the client application
 - The **Client secret** is an extra piece of information that authenticates the client application
 - The client secret plays a similar role as a password for the client
- When you create an API definition, API Connect creates a security definition for a Client ID



Example: Secure with Client ID (1 of 2)

- Select **Secure using Client ID** during API creation
 - The client ID API Key is generated into the security definitions for the API

The screenshot shows two panels side-by-side. On the left, the "Create New OpenAPI" screen has a "Secure" section where the "Secure using Client ID" checkbox is checked and highlighted with a yellow circle containing the number 1. On the right, a "Summary" panel lists two items: "Generated OpenAPI 2.0 definition" and "Applied security", each preceded by a green checkmark icon and a yellow circle containing the number 2.

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-10. Example: Secure with Client ID (1 of 2)

When the Secure using Client ID option is selected during the creation of the API, the client ID API Key type is generated into the security definitions for the API.

Example: Secure with Client ID (2 of 2)

- The client ID API Key type is added to the Security definitions

NAME	TYPE	LOCATED_IN
clientID	<input checked="" type="checkbox"/> apiKey	header

- Client ID API Key Security is enabled for the API

SECURITY DEFINITIONS
<input checked="" type="checkbox"/> clientID apiKey

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-11. Example: Secure with Client ID (2 of 2)

When the option that is named Secure using Client ID is selected during the creation of the API, the client ID API Key type is generated into the security definitions for the API.

The client ID API Key is also enabled on the security tab for the API.



Example: Add client secret security definition

- Manually add the client secret API Key security definition

API Setup

Security Definitions

Security definitions control client access to API endpoints, including API key validation, application user authentication, and OAuth. Learn more

Add

NAME	TYPE	LOCATED_IN
clientID	apiKey	header

Security Definitions

Security definitions control client access to API endpoints, including API key validation, application user authentication, and OAuth. Learn more

Add

NAME	TYPE	LOCATED_IN
Client Secret	apiKey	header
clientID	apiKey	header

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-12. Example: Add client secret security definition

You can add the client secret API Key security definition after you create the client ID API Key security definition.

Click the option to add a security definition. Type the name for the client secret.

Select API Key for the type and Client Secret for the key type.

The security definition is added when you save the changes.

Applying security definitions (1 of 2)

- You must enable the security definition to apply the security scheme in your API
 - Select the security definition option in the **Security** section of your API definition

The screenshot shows the 'Security Definitions' section of an API definition. On the left, there's a sidebar with tabs: API Setup, Security Definitions (selected), Security (highlighted in blue), Paths, Definitions, Properties, and Target. The main area has a title 'SECURITY DEFINITIONS'. It lists two items: 'Client Secret apiKey' (with an empty checkbox) and 'clientID apiKey' (with a checked checkbox). There's also an 'Add' button in the top right corner.

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-13. Applying security definitions (1 of 2)

After you create an API key security definition, you must apply the security requirement in the **Security** section of your API definition.

In the example, you added the client secret to the security definitions.

Select the security definition option in the security section of your API definition to apply the security scheme to your API.

Applying security definitions (2 of 2)

- When you enable a security definition, you automatically enable the setting to every operation in your API
- You can override this behavior by selecting a specific API operation
 - After you apply a security definition, you can enable or clear the security requirement on an API operation level in the **paths** API definition section



/sample

Operation

Operation Id (optional)

Summary (optional)

Tags (optional)

Description (optional)

Override API Security Definitions

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-14. Applying security definitions (2 of 2)

After you create an API key security definition, you must apply the security requirement in the **security** section of your API definition.

When you enable a security definition, you automatically enable the setting to every operation in your API. You can override this behavior by selecting a specific API operation, and changing the security setting in the API operation.

Rules for defining client ID and client secret

1. You can define at most one client ID and one client secret security definition in an API definition
2. For any API definition, you can apply:
 - **No API key** security definitions
 - One **client ID** security definition
 - One **client ID** and one **client secret** definition
3. If you require the application developer to supply both client ID and client secret, you must apply two separate API key security definitions
4. You can specify the client ID and client secret values as HTTP **headers or query parameters**
 - You must specify the same location for the client ID and client secret, either the header or query parameters

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-15. Rules for defining client ID and client secret

You cannot apply more than two API key security definitions to an API.

If you apply an API key security definition for client secret, you must also apply an API key security definition for client ID.

If you require the application developer to supply both client ID and client secret, you must apply two separate API key security definitions.

The API keys are sent in the request header or as a query parameter. Both API keys must specify the same location, either the header or query parameters.

Example: Client ID and client secret in the message header

- Two locations are used to store the client ID and client secret:
 - As **HTTP headers**:

```
GET https://localhost:4002/api/products
Content-Type: application/json
Accept: application/json
X-IBM-Client-Id: b91e945a-21wf-4869-bb7bay130d
X-IBM-Client-Secret: n29ax9RMn3ai2iasdf92DKSF92asdf
```

- As **query parameters**:

```
GET https://localhost:4002/api/products?client_id=
b91e945a-21wf-4869-bb7bay130d
&client_secret=n29ax9RMn3ai2iasdf92DKSF92asdf
Content-Type: application/json
Accept: application/json
```

Figure 10-16. Example: Client ID and client secret in the message header

The names of the HTTP headers and the query parameters are the default names that API Connect sets. You can change the name of these fields in the API key security definitions.

As the API developer, you choose whether to store API key information as headers or query parameters. The logical place to put client metadata is in the request message header. However, if you want to test a simple GET operation, it is easier to specify the client ID and client secret information as query parameters.

10.3. Authenticate clients with HTTP basic authentication

Authenticate clients with HTTP basic authentication

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-17. Authenticate clients with HTTP basic authentication

Topics

- API security concepts
- Identify client applications with API key
-  Authenticate clients with HTTP basic authentication
- Introduction to OAuth 2.0

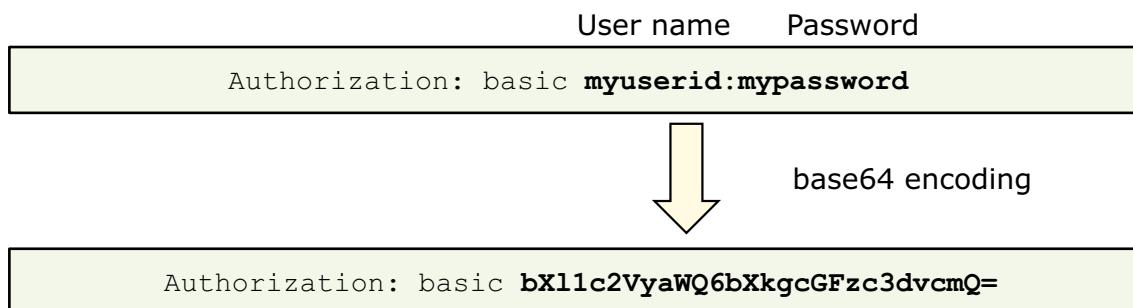
Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-18. Topics

Verifying identity with HTTP basic authentication

- **HTTP basic authentication** is a widely implemented scheme for sending client user credentials to a web server
 - The client writes the user name and password in the HTTP header
- User credentials are not encrypted or hashed in the header
 - Base64 encoding prevents sensitive data from being displayed as plain text when the message is transmitted
 - Base64 encoding does not protect the contents of the message from being intercepted and decoded with a Base64 decoder



[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-19. Verifying identity with HTTP basic authentication

The client writes the HTTP basic authentication information in the HTTP request message header. The name of the header is `Authorization`, followed by the keyword `basic`. The user name and password are separated with a colon. Before the client sends the request message, it encodes the user name, colon, and password with the base64 encoding scheme.

Keep in mind that this encoding scheme is not encryption – anyone who intercepts the message can decode the message and retrieve the user name and password.

Base64 is a scheme for encoding binary data as text. The most common use of Base64 is to encode photo, video, and document attachments to email.

Example: Storing credentials in HTTP request header

```

PUT /api/employee HTTP/1.1
Host: www.example.com/hr/
Authorization: basic bXl1c2VyaWQ6bdvcmQ=
Date: Mon, 12 Dec 2016, 15:35:12 GMT

{
  "fname" : "John",
  "lname" : "Smith",
  "email" : "jsmith@example.com",
  "dept" : "finance",
  "country" : "Canada"
}

```

The diagram illustrates the structure of the HTTP request message. Braces on the right side group components:

- A brace groups the **Authorization** header as the **HTTP basic authorization header**.
- A brace groups the **Host** and **Date** headers as the **HTTP header**.
- A brace groups the JSON data in the message body as the **HTTP body**.

Declaring client authorization requirements

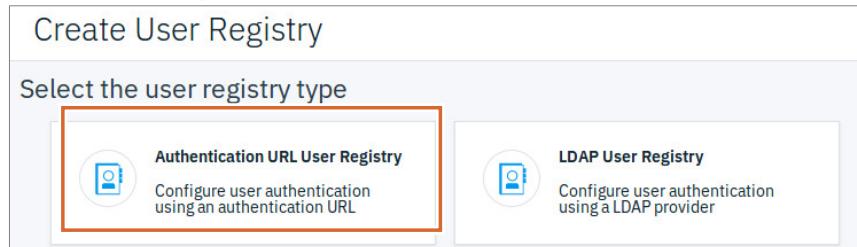
© Copyright IBM Corporation 2020

Figure 10-20. Example: Storing credentials in HTTP request header

The HTTP basic authentication header appears in the start of the request message. The data in the HTTP message body is specific to the web service operation. The service provider does not use the message body data during HTTP basic authentication.

Setting up a user registry (1 of 3)

- Before you can create a basic authentication security definition for an API, the user registry must exist
- API Connect supports three types of user registries:
 - Authentication URL user registry
 - LDAP user registry
 - Local user registry
- To create a user registry, you can use either API Manager or Cloud Manager
 - A registry that is created in API Manager is visible only to your provider organization
 - When you create a registry in Cloud Manager, you can make it visible to multiple provider organizations



[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-21. Setting up a user registry (1 of 3)

Before you set up a basic authentication security definition, you need to configure a user registry. API Connect supports three types of user registries: Authentication URL user registry, LDAP user registry, and Local user registry.

To create a user registry, you can use either API Manager or Cloud Manager.

When you create a registry in API Manager, it is visible only to your provider organization. When you create a registry in Cloud Manager, you can make it visible to multiple provider organizations.

To set up a user registry in API Manager, click the Resources option from the navigation menu.

Select User Registries. Then, select create.

In this example, the authentication URL user registry is selected.

You can also authenticate the client user name and password with an LDAP user registry. Specify the name of the user registry profile in this field. You must set up the LDAP user registry profile separately with the API Manager web interface.

Setting up a user registry (2 of 3)

The screenshot shows the 'Authentication URL User Registry' configuration page. It includes fields for Title, Name, Summary (optional), URL, and TLS Client Profile (optional). Three yellow circles with numbers 1, 2, and 3 are overlaid on the page, pointing to the 'Title' field, the 'URL' field, and the 'TLS Client Profile' dropdown respectively.

Authentication URL User Registry	
Title	BasicAuthURL
Name	basicauthurl
Summary (optional)	
URL	https://services.think.ibm:1443/authorize
TLS Client Profile (optional)	No TLS Profile

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-22. Setting up a user registry (2 of 3)

The page shows the create user registry page in the API Manager web application

1. Type a unique title and name for your authentication URL user registry.
2. Type the network endpoint for the authentication service. If the client sent a valid user name and password, the authentication service returns an HTTP status code of 200 OK. Otherwise, the service returns a 401 Unauthorized status code.
3. Optionally, enter the name of a **TLS profile**. The Transport Layer Security (TLS) profile contains the settings and certificates that the API gateway uses to set up an HTTPS connection to the client application. You must set up a TLS profile separately with the API Manager web interface.

The user registry is added to the registries on the resources page in API Manager.



Setting up a user registry (3 of 3)

- Configure the catalog to use the registry

A screenshot of a web-based interface titled "Manage / Sandbox Settings". On the left, there's a sidebar with links: Overview, API User Registries (which is highlighted in blue), Gateway Services, Lifecycle Approvals, Roles, and Onboarding. The main content area is titled "API User Registries" and contains a table with one row. The table has columns for "TITLE", "TYPE", and "SUMMARY". The single row shows "BasicAuthURL" in the TITLE column, "Authentication URL" in the TYPE column, and an empty SUMMARY field. A blue "Edit" button is located at the top right of the table area.

TITLE	TYPE	SUMMARY
BasicAuthURL	Authentication URL	

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-23. Setting up a user registry (3 of 3)

Associate the user registry to the catalog where the API is published.

You can now set basic authentication for an API and select the basic authentication URL registry.

Example: Basic authentication security definition

- Add basic authentication to the security schemes
 - The authentication URL can be selected

The screenshot shows the 'API Security Definition' page. On the left, there's a form with fields for 'Name' (set to 'basic auth'), 'Description (optional)', 'Type' (set to 'Basic'), and 'Authenticate using User Registry' (set to 'BasicAuthURL'). A yellow circle labeled '1' is over the 'Name' field. On the right, a sidebar has tabs for 'API Setup', 'Security Definitions' (which is selected), 'Paths', 'Definitions', and 'Properties'. The 'Security Definitions' tab displays a table of existing definitions:

NAME	TYPE	LOCATED_IN
basic auth	basic	:
clientIdHeader	apiKey	header

A yellow circle labeled '2' is over the 'Add' button in the top right corner of the sidebar.

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-24. Example: Basic authentication security definition

1. Add a security definition in API Manager.

Select the Basic type.

Select the Authentication URL user registry from the list under the heading that is named authenticate using user registry.

Save the change.

2. The basic authentication type is added to the security definitions.



Example: Apply basic authentication security to the API

- Select the basic authentication option in the Security tab for the API

A screenshot of the API Manager interface. On the left, there's a vertical navigation bar with options: API Setup, Security Definitions, Security (which is highlighted in blue), Paths, and Definitions. The main content area has a title "Security" and a sub-section "SECURITY DEFINITIONS". Below this, there's a list with a checked checkbox next to the text "basic auth basic". In the top right corner of the main area, there's a blue "Add" button.

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-25. Example: Apply basic authentication security to the API

Apply the basic authentication security to the API from the Security option in API Manager.

10.4. Introduction to OAuth 2.0

Introduction to OAuth 2.0

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-26. Introduction to OAuth 2.0

Topics

- API security concepts
 - Identify client applications with API key
 - Authenticate clients with HTTP basic authentication
-  Introduction to OAuth 2.0

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-27. Topics

What is OAuth?

- OAuth defines a way for a client to access server resources on behalf of another party
- It provides a way for users to authorize a third party to their server resources without sharing their credentials to a third-party application
- It separates the identity of the resource owner (user) from a third-party application that acts on behalf of the user
- The resource owner specifies which resources the OAuth client can access, and for how long

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-28. What is OAuth?

The OAuth specification solves a specific problem: how to delegate access rights to a third-party client that works on behalf of the user. Before OAuth, third-party applications would ask and store the user's user name and password within the application. This process is risky, as the server cannot distinguish between the user and the third-party application. One analogy in the real world is to hand over your house keys to a cleaning service. You must have a high degree of trust in the client to give them complete access to your home.

With OAuth, the client does not use your credentials. Instead, an authorization service gives a temporary pass to the client, so it can do a limited set of tasks in a fixed time period. As the user, you can tell the authorization service to revoke the temporary pass at any time.

Although OAuth is more complicated than handing over your credentials to the client, it is a safer mechanism that gives the user control over the third-party client's actions.



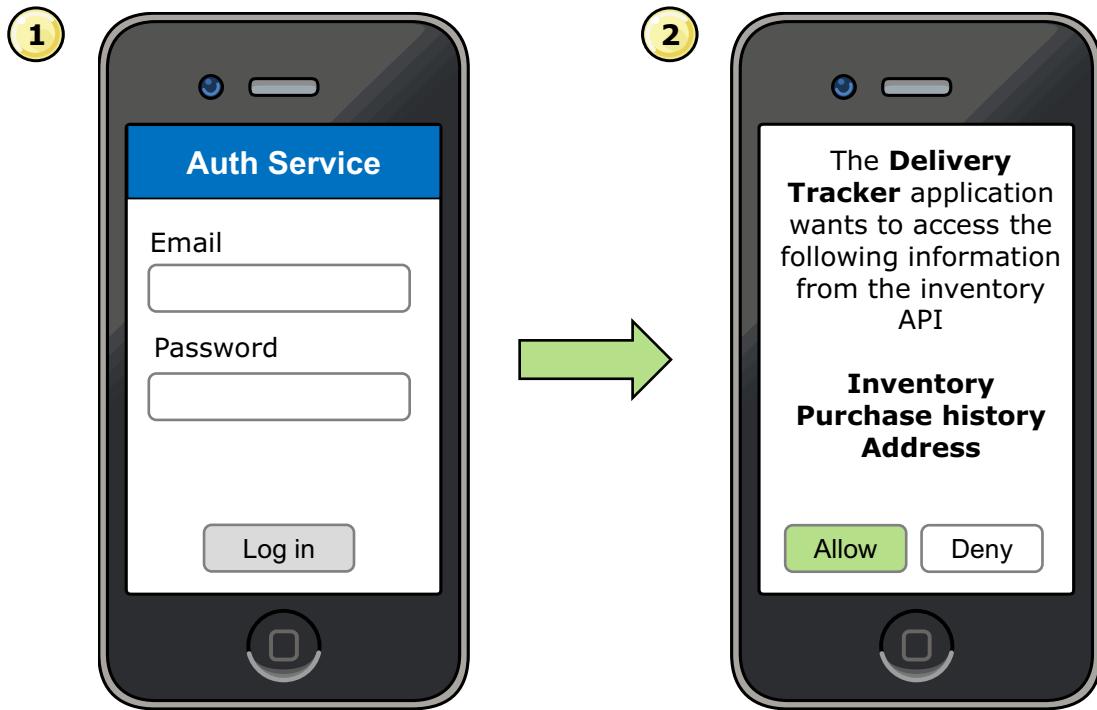
Information

OAuth separates the role of the client from the role of the resource owner.

The client is issued a different set of credentials than the credentials of the resource owner.

Instead of using the resource owner's credentials to access a protected resource, the client obtains an access token, which is a string that denotes a specific scope, lifetime, and other access attributes.

Example: Allow third-party access to shared resources



Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-29. Example: Allow third-party access to shared resources

Whenever you sign up for a web-based application or a mobile application, you create an account on the server with a user name and password. The process becomes tedious for users when they sign up for dozens of applications.

Social networks, such as Facebook and Twitter, already link your identity to a user account. Therefore, many applications use your social network account to create an account.

This scenario has four participants: you as the user, the third-party application as a client, the shared resources on the web, and the authentication service. You want the third-party application to access some (but not all) of your information from the service. That is, you want the client to act on your behalf to access resources on the service.

In this example, the third-party application, the Delivery Tracker, wants to access your product inventory records from the inventory API. The application opens a new page from the authorization service. After you log in and allow the application to access the information, the authorization service grants an access token to the application. At no time does the third-party application see your user name or password on the authorization service.

Example: Third-party access to inventory API resources

OAuth allows social network applications to share resources



- Alice is the **owner** of inventory records
- As a **resource owner**, Alice declares which applications can access the inventory API on her behalf



- The inventory API provides online access to inventory records
- The service that runs the API acts as the **resource service**
- It manages access to Alice's records



- Delivery tracker, a third-party **client application**, wants to access Alice's inventory records from the API

- An **authorization service** verifies the identity of the client that wants to access Alice's records
- This server issues a token or a code to access the inventory API from the resource server

[Declaring client authorization requirements](#)

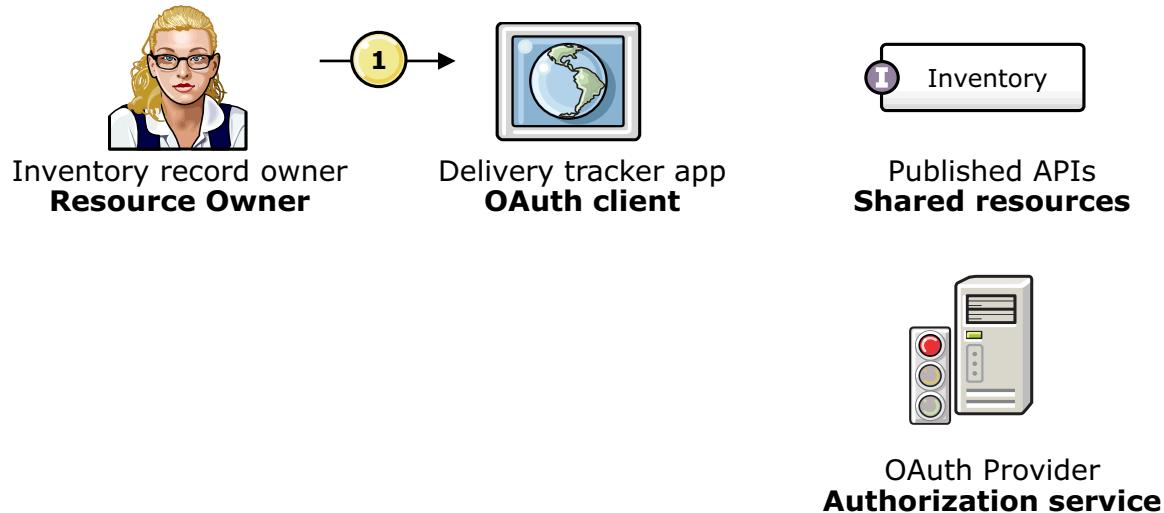
© Copyright IBM Corporation 2020

Figure 10-30. Example: Third-party access to inventory API resources

Take a closer look at the three actors in the OAuth scenario. Alice is the **owner** of inventory records. As the **user**, Alice wants to feed the current inventory records to a package tracker application. The Delivery Tracker is a **third-party client application** that wants to access Alice's inventory records. Last, the inventory API is a service that securely stores Alice's records. This service also manages access to the records from Alice and third-party applications that act on Alice's behalf.

OAuth Step 1: Resource owner requests access

- Step 1: Alice, as the resource owner, requests access to the inventory API from the client application, the delivery tracker app



[Declaring client authorization requirements](#)

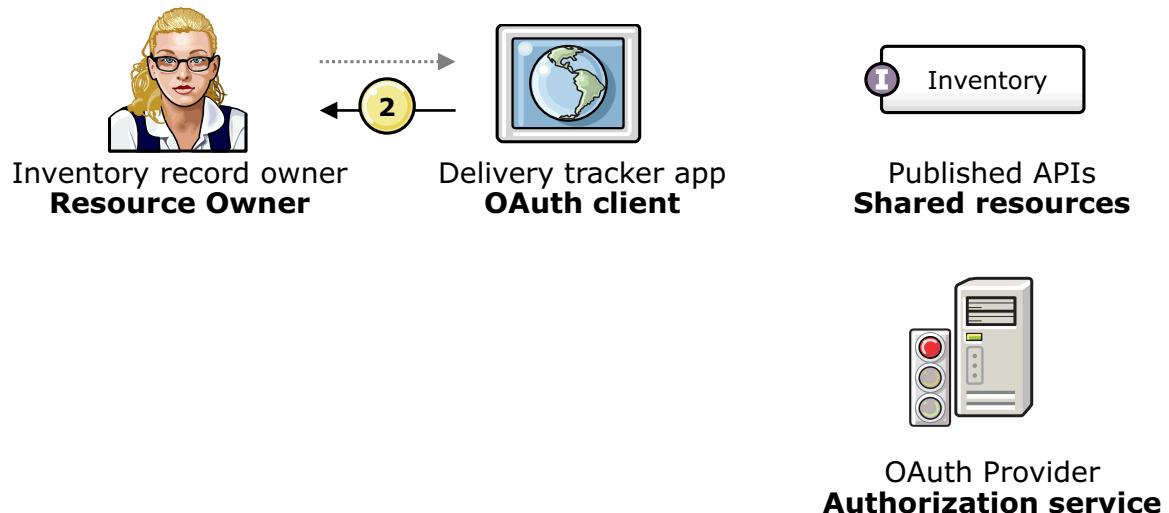
© Copyright IBM Corporation 2020

Figure 10-31. OAuth Step 1: Resource owner requests access

In this scenario, Alice is the owner of inventory records in the inventory API. Alice wants to track the delivery status of her purchases through the delivery tracker application. Alice is the resource owner, and the delivery tracker application is a OAuth client application. Alice starts the process when she selects the “look up your deliveries” option in the delivery tracker application.

OAuth Step 2: OAuth client redirection to owner

- Step 2: The OAuth client sends the resource owner a redirection to the authorization service



[Declaring client authorization requirements](#)

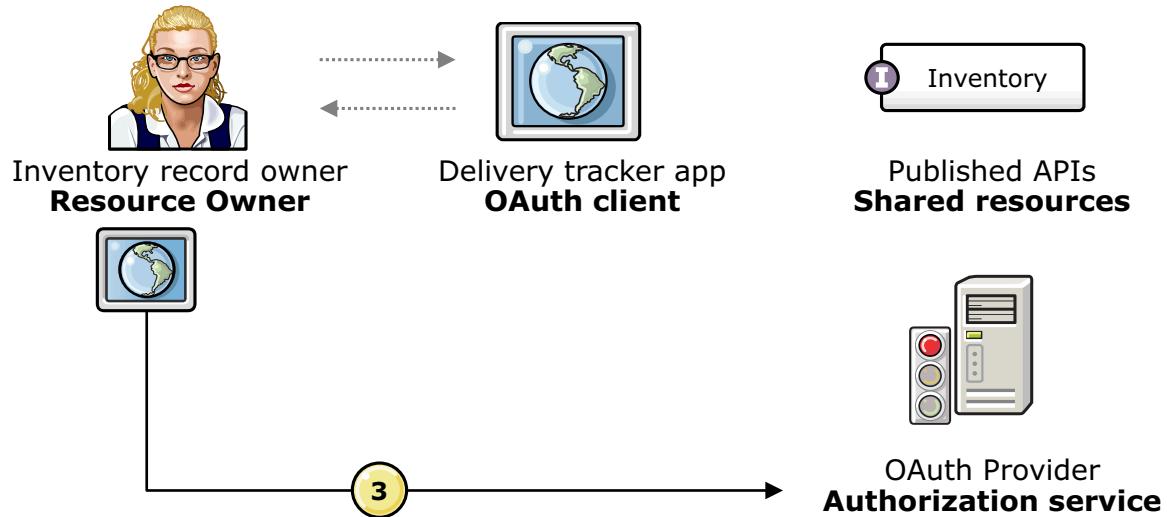
© Copyright IBM Corporation 2020

Figure 10-32. OAuth Step 2: OAuth client redirection to owner

In the second step, the delivery tracker application requires the resource owner's authorization before it can access the owner's inventory records. Instead of asking Alice directly for her user credentials, the client application redirects Alice's request to an authorization service.

OAuth Step 3: Authenticate owner with authorization service

- Step 3: The resource owner authenticates against the authorization service



[Declaring client authorization requirements](#)

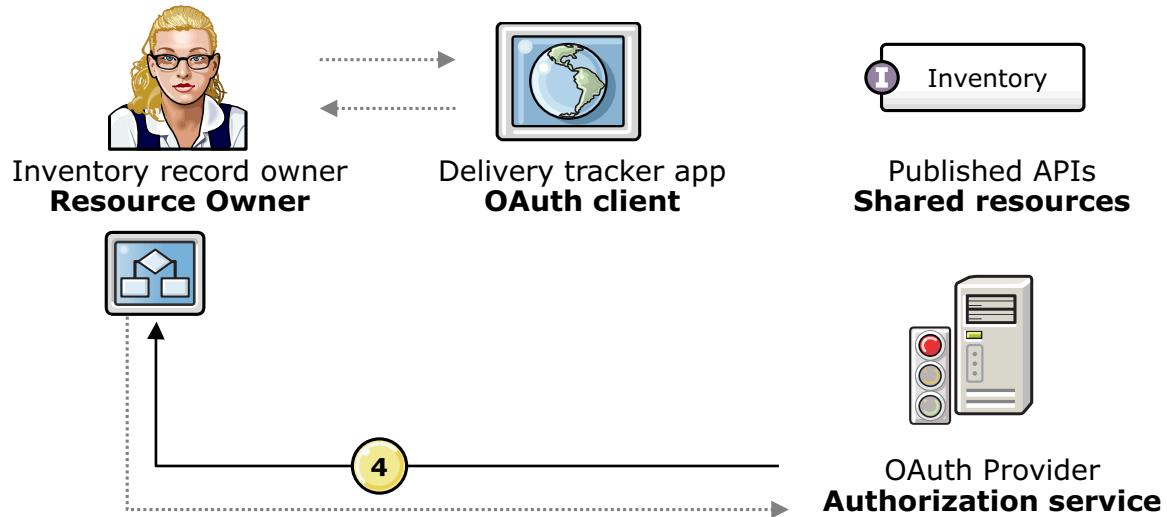
© Copyright IBM Corporation 2020

Figure 10-33. OAuth Step 3: Authenticate owner with authorization service

In the third step, the authorization server asks for Alice's user credentials to verify her identity.

OAuth Step 4: Ask resource owner to grant access to resources

- Step 4: The authorization service returns a web form to the resource owner to grant access



[Declaring client authorization requirements](#)

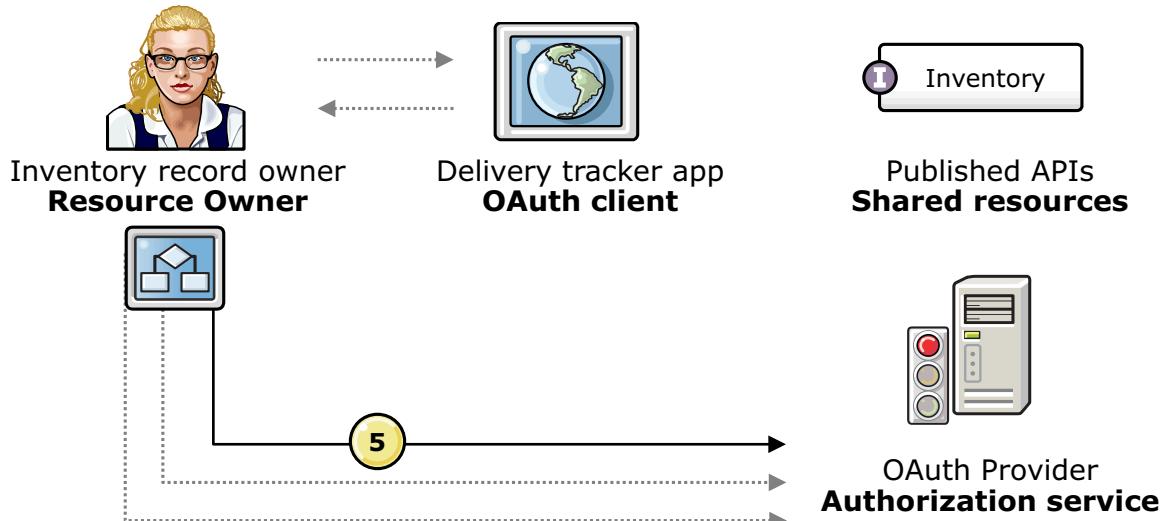
© Copyright IBM Corporation 2020

Figure 10-34. OAuth Step 4: Ask resource owner to grant access to resources

The authorization service returns a web form to ask Alice whether she grants the OAuth client access to her resources. That is, does the delivery tracker application have permission to look up inventory records from the API, on Alice's behalf?

OAuth Step 5: Resource owner grants client access to resources

- Step 5: The resource owner submits the form to allow or to deny access



[Declaring client authorization requirements](#)

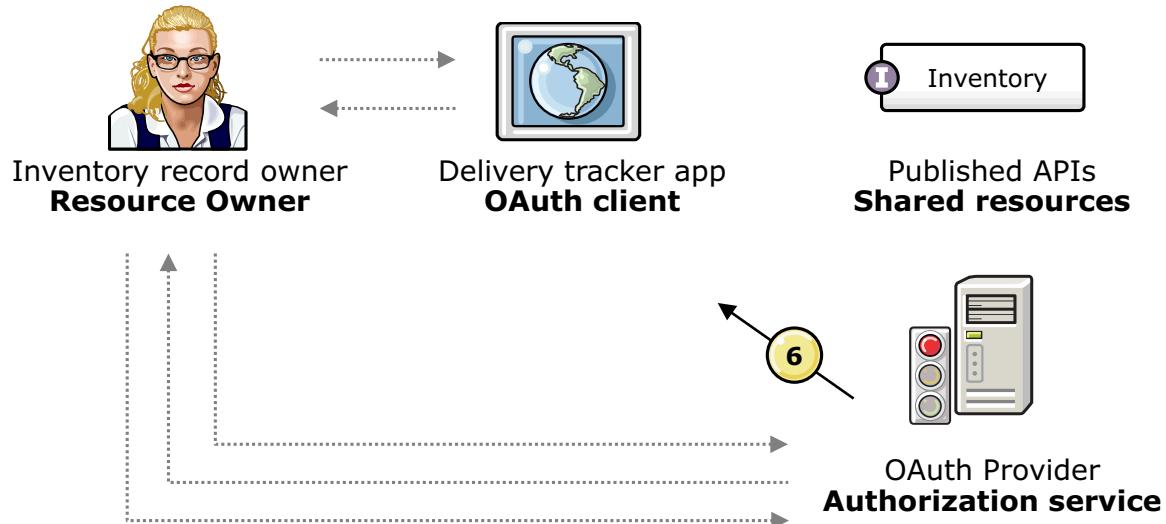
© Copyright IBM Corporation 2020

Figure 10-35. OAuth Step 5: Resource owner grants client access to resources

The resource owner, Alice, submits the web form to allow or deny access to her resources.

OAuth Step 6: Authorization service sends authorization grant code to client

- Step 6: If the resource owner allows access, the authorization service sends the OAuth client a redirection with the authorization grant code



Declaring client authorization requirements

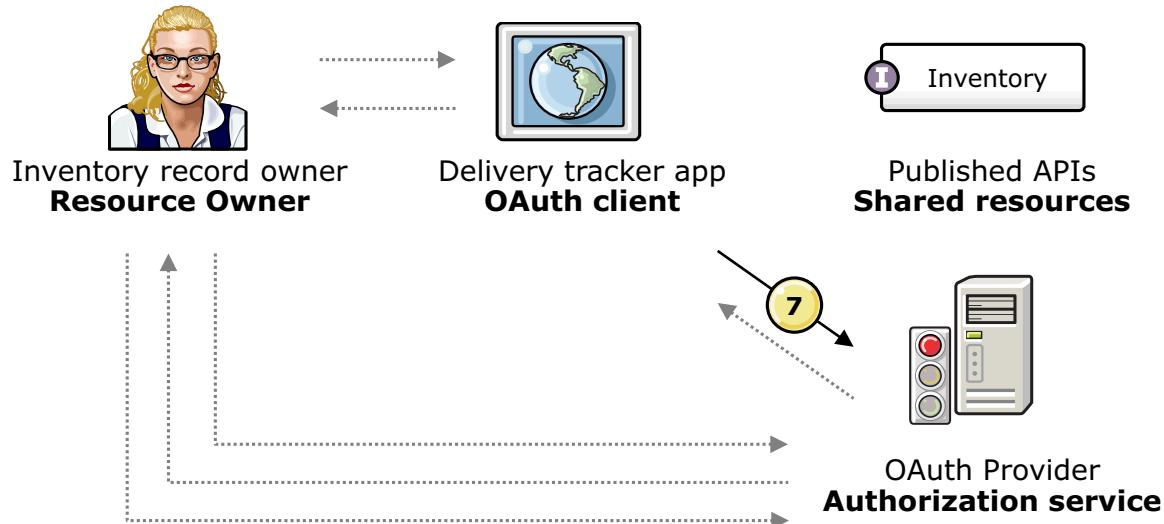
© Copyright IBM Corporation 2020

Figure 10-36. OAuth Step 6: Authorization service sends authorization grant code to client

The authorization service never transmits the resource owner's user name and password to the OAuth client. Instead, the service sends an authorization grant code: a token that the OAuth client can use to access Alice's resources on her behalf.

OAuth Step 7: Client requests access token from authorization service

- Step 7: To access the resource, the OAuth client sends the authorization grant code and other information to the authorization service



[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-37. OAuth Step 7: Client requests access token from authorization service

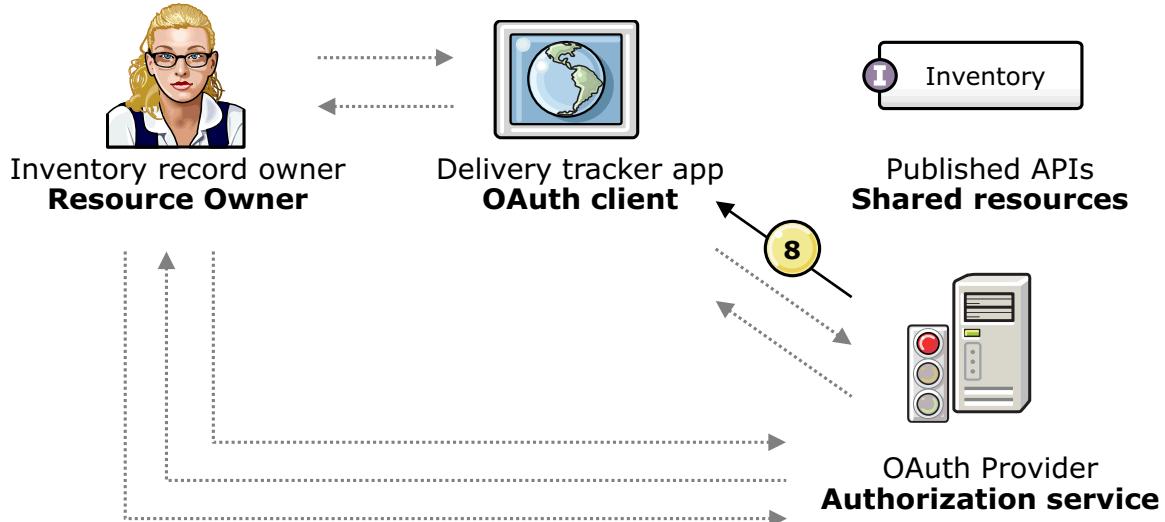
The OAuth client sends three pieces of information to the authorization service: an authorization grant code, the client ID, and the client secret.

The API Key (client ID and client secret) identifies the calling application to the resource service that runs the APIs.

The Delivery tracker application needs to register with the authorization service, during the setup of the OAuth process, before any OAuth requests are made.

OAuth Step 8: Authorization server sends authorization token to client

- Step 8: If the authorization service verifies the grant authorization information, it returns an access token to the OAuth client



Declaring client authorization requirements

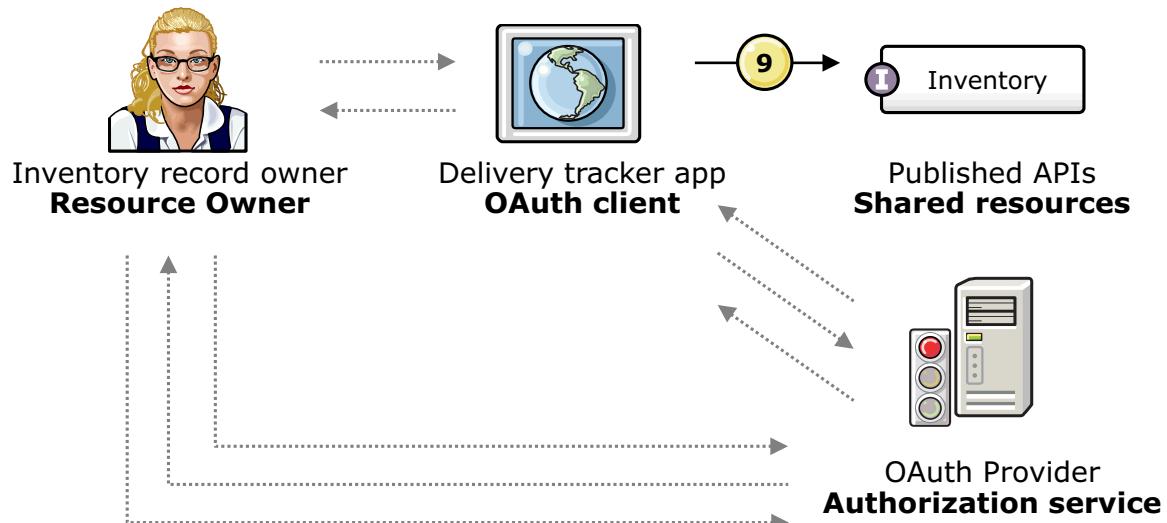
© Copyright IBM Corporation 2020

Figure 10-38. OAuth Step 8: Authorization server sends authorization token to client

The authorization service verifies the grant authorization information. Then, it returns an access token to the OAuth client.

OAuth Step 9: OAuth client sends access token to resource service

- Step 9: The OAuth client sends the access token to the resource service



Declaring client authorization requirements

© Copyright IBM Corporation 2020

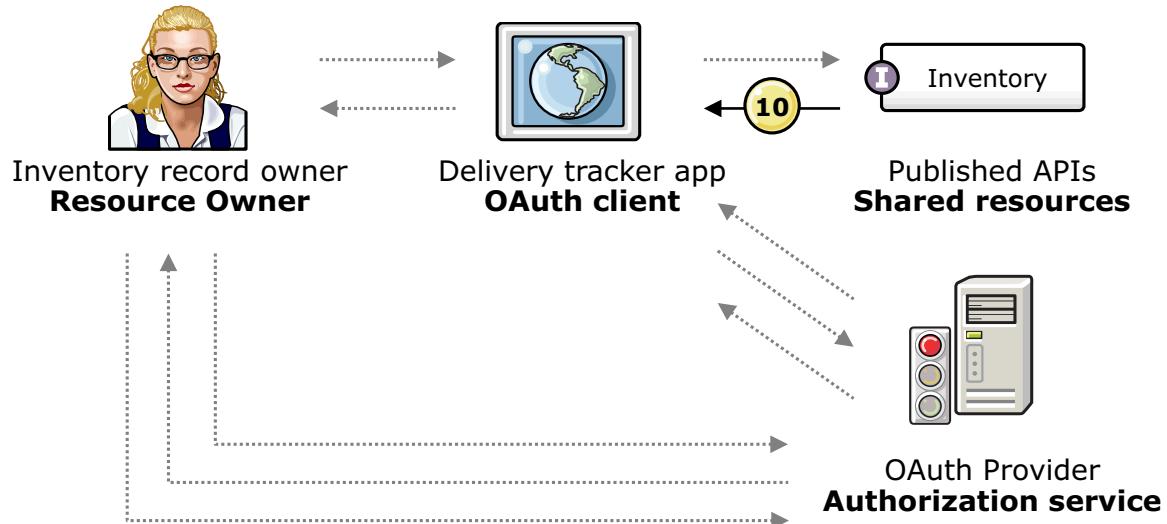
Figure 10-39. OAuth Step 9: OAuth client sends access token to resource service

The client application sends the access token to the resource service.

The authorization service and the resource service can run on the same server.

OAuth Step 10: Resource server grants access to OAuth client

- Step 10: If the access token is valid for the requested resource, the resource server allows the OAuth client to access the resource



[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2020

Figure 10-40. OAuth Step 10: Resource server grants access to OAuth client

If the access token is valid for the requested resource, the resource service allows the OAuth client to access the resources. Only the APIs that were allowed by the resource owner can be accessed by the delivery tracker application.

Unit summary

- Identify the security definition options in API Connect
- Explain the concept and use cases for API keys
- Explain the concept and use cases for HTTP basic authentication
- Explain the concept and use cases for OAuth 2.0 authorization
- Explain the steps in the OAuth 2.0 message flow

Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-41. Unit summary

Review questions

1. Which one of the following sentences best describe the client ID?
 - A. The client ID represents the person who signs on to the web application.
 - B. The client ID represents the client application.
 - C. The client ID represents the client application developer.
 - D. The client ID represents the resource owner.

2. What is the purpose of an API key?
 - A. The API key scheme enforces role-based access to API products.
 - B. The API key scheme authenticates the API caller.
 - C. The API key scheme defines the API plan.
 - D. The API key scheme secures API traffic.



Declaring client authorization requirements

© Copyright IBM Corporation 2020

Figure 10-42. Review questions

Write your answers here:

1.

Review answers



1. Which one of the following sentences best describe the client ID?
 - A. The client ID represents the person who signs on to the web application.
 - B. The client ID represents the client application.
 - C. The client ID represents the client application developer.
 - D. The client ID represents the resource owner.

The answer is B.

2. What is the purpose of an API key?
 - A. The API key scheme enforces role-based access to API products.
 - B. The API key scheme authenticates the API caller.
 - C. The API key scheme defines the API plan.
 - D. The API key scheme secures API traffic

The answer is B.

Unit 11. Creating an OAuth 2.0 provider

Estimated time

00:45

Overview

This unit examines the OAuth 2.0 provider. In an OAuth 2.0 message flow, the OAuth provider is an authorization server that issues access tokens to authorized clients. In an API Connect cloud, you can configure the API gateway to act as an OAuth 2.0 Provider. This unit explains how to create and configure a Native OAuth Provider in either the Cloud Manager or API Manager graphical applications.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Explain the concept of an OAuth provider
- Describe the steps to secure an API with OAuth 2.0
- Identify the OAuth Provider types
- Explain how to create a Native OAuth Provider
- Explain the OAuth flow and grant types
- Explain the difference between public and confidential schemes
- Describe how to configure security settings for an API

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-1. Unit objectives

Role of IBM API Connect in the OAuth flow

- **OAuth** is a token-based authorization protocol that allows other applications or websites access to data without requiring the user to share personal information
- In an OAuth flow, IBM API Connect hosts APIs as **shared resources** and provides the **authorization and token services**



Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-2. Role of IBM API Connect in the OAuth flow

OAuth is an open-standard authorization protocol that provides applications or websites the ability for secure designated access.

In IBM API Connect, the OAuth Provider implements the **authorization** and **token** services in an OAuth flow. If you already have an OAuth Provider, you can configure the OAuth 2.0 security definition to call your existing authorization service instead.

The OAuth specification is designed for use with HTTP.

What are the steps to secure an API with OAuth 2.0?

- To secure your API with OAuth 2.0, you must configure two services in IBM API Connect:

1. Configure an OAuth Provider

- The **OAuth Provider** is a security service that provides the **token** and **authorize** API operations
- You configure the settings for the OAuth 2.0 Provider in the Resources page in either Cloud Manager or API Manager

1. For each API that you want to secure, declare an **OAuth 2.0 security definition**

- You declare which API operations you want to secure at the API Gateway
- You specify how the gateway handles authentication, authorization, and token management tasks

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-3. What are the steps to secure an API with OAuth 2.0?

To secure your API with OAuth 2.0, you must configure two services in IBM API Connect:

Create an OAuth 2.0 Provider.

The OAuth 2.0 Provider is a security service that provides the token and authorize API operations.

You configure the settings for the OAuth 2.0 Provider in the Resources page in either Cloud Manager or API Manager. When you configure the OAuth Provider in Cloud Manager, the OAuth Provider can be applied to all provider organizations. When configured in API Manager, the OAuth Provider is visible to or only selected provider organizations.

The API gateway implements the OAuth 2.0 Provider API operations

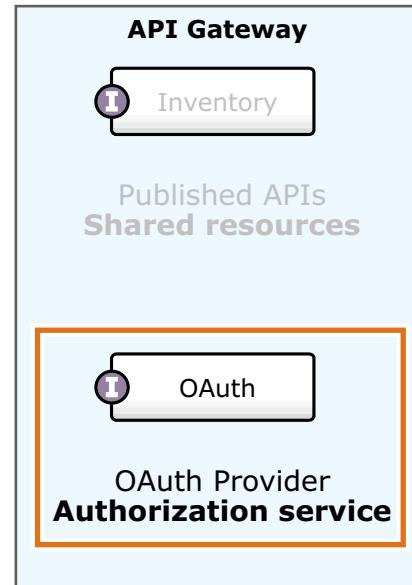
For each API that you want to secure, declare an OAuth 2.0 security definition

You declare which API operations you want to secure at the API Gateway

You specify how the gateway handles authentication, authorization, and token management tasks

What is an OAuth Provider?

- The OAuth Provider is a security service that authorizes access to API operations
- The OAuth 2.0 specification defines two REST API operations:
 - The **/authorize** operation reads the client credentials and the requested resource, and determines whether to grant access to the API client
 - The **/token** service takes an **authorization grant code**, and returns an **access token**: a time-limited permission to call an API operation on the server



[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-4. What is an OAuth Provider?

The OAuth Provider is a security service that authorizes access to API operations.

Instead of sharing passwords, OAuth uses authorization tokens to verify the identity between clients and service providers.

11.1. Create an OAuth Provider



Figure 11-5. Create an OAuth Provider

Topics

Create an OAuth Provider

- Secure an API with OAuth 2.0 authorization

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-6. Topics

OAuth Provider types

API Connect supports two OAuth Provider types:

- **Native OAuth Provider**

- Configured and managed by you within your cloud
- A Native OAuth Provider object provides settings for OAuth processing:
 - Extract the application user's credentials
 - Authenticate their identity
 - Grant authorization
 - Generate and validate OAuth tokens
- The API gateway implements the OAuth 2.0 Provider API operations

- **Third-party OAuth Provider**

- Configure the secure endpoints to provide OAuth authentication from a third party

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-7. OAuth Provider types

You can configure two types of OAuth providers in API Connect:

- Native OAuth providers are configured and managed by you within your cloud.
- Third-party OAuth providers are configured by typing the secure endpoints to provide OAuth authentication from a third party.

The examples that follow show how to configure a Native OAuth Provider.



Create an authentication registry (1 of 3)

- Create an authentication registry in Cloud Manager or API Manager

The screenshot shows the 'User Registries' section in the Cloud Manager interface. A 'Create' button is highlighted with a red box. Below it, two options are shown: 'Authentication URL User Registry' and 'LDAP User Registry'. The 'Authentication URL User Registry' option is also highlighted with a red box.

TITLE	TYPE	VISIBLE TO
API Manager Local User Registry	Local User Registry	Private
Cloud Manager Local User Registry	Local User Registry	Private

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-8. Create an authentication registry (1 of 3)

When you configure a Native OAuth Provider, you must first configure the authentication registry that is used to extract the user credentials and authenticate their identities.

User authentication is required for the implicit and access code (authorization code) grant types.

You can create the authentication registry either in Cloud Manager or API Manager.

When you create the authentication registry in Cloud Manager, the registry can be used by multiple provider organizations.

In the example, an authentication URL user registry is selected.

Create an authentication registry (2 of 3)

- Type the title, name, and URL for the authentication service
- API Connect calls the authentication service later with the user name and password
- Either 200 OK or 401 Unauthorized is returned

Authentication URL User Registry	
Title	AuthURLRegistry
Name	authurlregistry
Summary (optional)	
URL	https://httpbin.org/basic-auth/user/pass
Please select a TLS Client Profile (optional)	
No TLS Profile	

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-9. Create an authentication registry (2 of 3)

An Authentication URL user registry provides a simple mechanism for authenticating users by referencing a custom identity provider.

Type the URL for the authentication service. API Connect makes a GET call to the URL to initiate the authentication process. The call includes the user name and password is collected from the user in its authorization header. Either 200 OK or 401 Unauthorized is returned.

Create an authentication registry (3 of 3)

- The authentication URL registry is now a public registry in Cloud Manager

User Registries			Create	⋮
<input type="checkbox"/> TITLE	TYPE	VISIBLE TO		⋮
<input type="checkbox"/> API Manager Local User Registry	Local User Registry	Private	⋮	⋮
<input type="checkbox"/> AuthURLRegistry	Authentication URL	Public	⋮	⋮
<input type="checkbox"/> Cloud Manager Local User Registry	Local User Registry	Private	⋮	⋮

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-10. Create an authentication registry (3 of 3)

The registry is added as a registry in Cloud Manager or API Manager and can be referenced by the provider organization.

Create a Native OAuth Provider (1 of 5)

- Create a Native OAuth Provider from the Resources page in Cloud Manager or API Manager

The screenshot shows the 'Resources' page in the 'Cloud Manager' section of the 'IBM API Connect' interface. On the left, there's a sidebar with icons for User Registries, OAuth Providers (which is selected and highlighted in blue), TLS, and Notifications. The main area is titled 'OAuth Providers' and contains a table with columns: TITLE, TYPE, and VISIBLE TO. There is one row visible in the table, labeled 'Native OAuth provider'. A tooltip for this row indicates it is a 'Native OAuth provider'. Below the table, a message says 'No items found'.

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-11. Create a Native OAuth Provider (1 of 5)

The next sequence of pages provides an example of how to configure a Native OAuth Provider in Cloud Manager or API Manager. From the Resources page, select the OAuth Providers option. Click the Add icon. Then, select Native OAuth provider from the list.

Create a Native OAuth Provider (2 of 5)

- Type the title and select the gateway type

Native OAuth Provider

Title
MainProviderOA

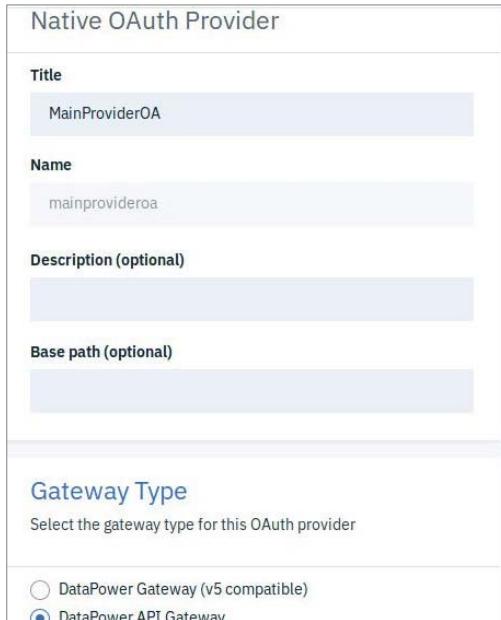
Name
mainprovideroa

Description (optional)

Base path (optional)

Gateway Type
Select the gateway type for this OAuth provider

DataPower Gateway (v5 compatible)
 DataPower API Gateway



[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-12. Create a Native OAuth Provider (2 of 5)

On the first page of the create Native OAuth Provider, you type the title and name. You must also select the gateway type. You are prompted to click Next to go to the following page.



Create a Native OAuth Provider (3 of 5)

- Select access code and resource owner password grant types

Configuration

Authorize path
/oauth2/authorize

Token path
/oauth2/token

Supported grant types

Implicit
 Application
 Access code
 Resource owner password

Supported client types

Confidential
 Public

Back **Cancel** **Next**

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-13. Create a Native OAuth Provider (3 of 5)

In the **supported grant types** section, select which of the four OAuth flows you want to use. An OAuth flow is the procedure that client applications follow to request access to a shared resource. A later slide explains these grant types.

Select either a **public** or **confidential** OAuth client type. A later slide explains the implications of each client type.

Create a Native OAuth Provider (4 of 5)

- Type the scope string for the OAuth Provider
 - When the client application requests an access token, it can specify an access scope
 - Scopes provide a way to limit the access that is granted in an access token
 - The access token that is issued to the application is limited to the scopes that are granted
 - Respond only to API requests from tokens that contain this scope

NAME	DESCRIPTION	DELETE
scope1	Sample scope description	

Scopes
Add scopes for this OAuth Provider.

Add

NAME DESCRIPTION DELETE

scope1 Sample scope description

Back Cancel Next

[Creating an OAuth 2.0 provider](#)

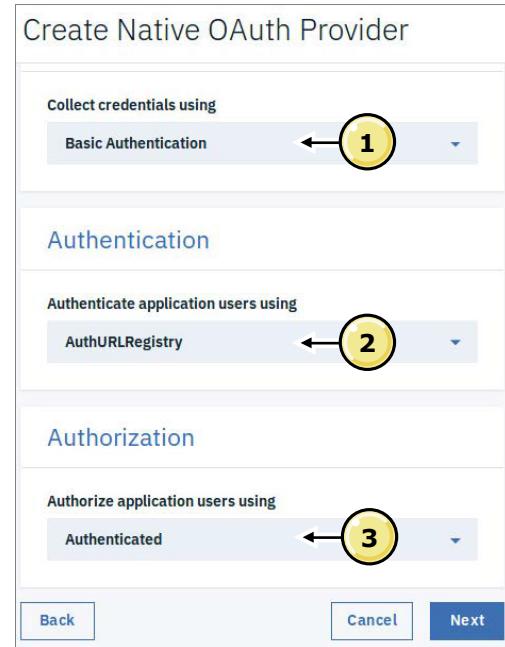
© Copyright IBM Corporation 2020

Figure 11-14. Create a Native OAuth Provider (4 of 5)

Create a **scope** for the OAuth provider. When the client application requests an access token, it can specify an access scope. For example, you can create two scopes for your API: one that authorizes read access to an account balance, and one that authorizes updates to the account balance with fund transfers.

Create a Native OAuth Provider (5 of 5)

1. To authenticate the client, the authorization service must **extract** the client identity
 2. The **Authentication** setting determines how the authorization service verifies the identity of the client
 3. The **Authorization** setting determines how the OAuth provider authorizes access to resources
- In the example, these values are selected:
 - Collect credentials: Basic Authentication
 - Authenticate users: AuthURLRegistry
 - Authorize users: Authenticated



Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-15. Create a Native OAuth Provider (5 of 5)

Configure the settings for collect credentials, authenticate users, and authorize users.

In the example, the authentication URL registry that was created earlier is selected for the authenticate users settings.

1. To authenticate the client, the authorization service must extract the client identity. You can retrieve client credentials from the context variable of a web form or from the HTTP Basic authentication header.
2. The **Authentication** setting determines how the authorization service verifies the identity of the client. In this example, the OAuth Provider sends the extracted client identity to the authentication URL. If the service at the authentication URL returns a status code of 200, the OAuth provider proceeds to the next step in the OAuth flow.
3. The **Authorization** setting determines how the OAuth provider authorizes access to resources. The **Authenticated** setting automatically grants access to any authenticated client. You can set this value to disabled.



Native OAuth Provider in Cloud Manager resources

- The Native OAuth Provider is added in Cloud Manager and is visible to public

The screenshot shows the "Cloud Manager" section of the IBM API Connect interface. On the left, there's a sidebar with icons for User Registries, TLS, and OAuth Providers (which is highlighted in blue). The main area is titled "Resources" and contains a table for "OAuth Providers". The table has columns for "TITLE", "TYPE", and "VISIBLE TO". One row is visible, showing "MainProviderOA" as the title, "Native" as the type, and "public" as the visible to value. There's also a "More" button (three dots) next to the row. A blue "Add ▾" button is located at the top right of the table.

	TITLE	TYPE	VISIBLE TO	⋮
	MainProviderOA	Native	public	⋮

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-16. Native OAuth Provider in Cloud Manager resources

The Native OAuth Provider is added in Cloud Manager as a shared service that is visible to all provider organizations. You are still required to select this OAuth Provider before you can use it in your catalog.

OAuth Provider: OAuth flow and grant types

OAuth flow OAuth grant type and explanation

Implicit	<ul style="list-style-type: none"> ▪ Uses an implicit grant type ▪ The authorization server sends back an access token after the resource owner authorizes the client application to use the resource
Password	<ul style="list-style-type: none"> ▪ Uses the resource owner password credentials ▪ The client application sends the user name and password for a user on the resource server
Application	<ul style="list-style-type: none"> ▪ Uses the client credentials ▪ The client application sends its own credentials when it accesses resources under its own control, or previously arranged with the authorization server
Access code	<ul style="list-style-type: none"> ▪ After the authorization server authenticates the resource owner, the authentication server sends back a custom redirect URI and an authorization code ▪ The client application opens the redirect URI with the authorization code to retrieve an access token for a resource

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-17. OAuth Provider: OAuth flow and grant types

OAuth 2.0 authorization grant types have four options:

- With the authorization code, the authorization server sends back a custom redirect URI and an authorization code after it authenticates the resource owner. The authorization code prevents replay attacks.

The client application opens the redirect URI with the authorization code to retrieve an access token for a resource.

- With the implicit grant type, the authorization server does not send back an authorization code. It sends back an access token after the resource owner authorizes the client application. This grant type is available for public clients only.
- With the resource owner password credentials grant type, the client application sends the user name and password for a user on the resource server. This grant type assumes a high level of trust between the client application and the resource server.
- With the client credentials grant type, the client application sends its own credentials when it accesses server resources under its own control, or resources that are previously arranged with the resource server. This grant type is available to confidential client types only.

OAuth Provider: Client types

- The OAuth 2.0 specification defines two types of clients:
 - Public
 - Confidential
- **Public clients** cannot be trusted with password secrets
 - For example, a web application that is written in JavaScript that runs on the user's web browser cannot ensure password confidentiality
- **Confidential clients** can keep a client password secret
 - The same web application that runs in an access-restricted web server keeps the password encrypted when it communicates with the server

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

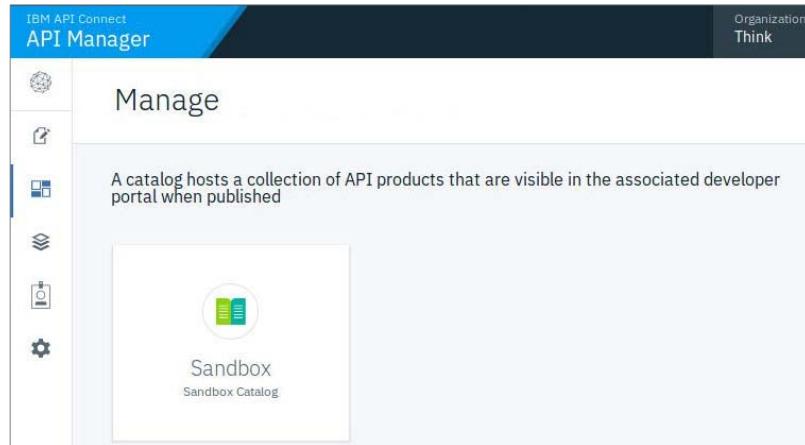
Figure 11-18. OAuth Provider: Client types

The client type setting defines whether the client application can keep a client password secret. For example, an access-restricted web server hosts a web application. Nobody except the system administrator can access the server and see the client password. This scenario is an example of a confidential client.

If the same web application runs as a JavaScript application on a web browser, a malicious user can break into the application and retrieve the password. In this case, the application is considered a public client because it cannot ensure that it can keep the client password confidential.

Configure the catalog to use the resources (1 of 2)

- Sign on to API Manager as the owner of the provider organization
- Go to the catalog that you want to use



[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-19. Configure the catalog to use the resources (1 of 2)

To use the Native OAuth Provider that is created in Cloud Manager, sign on to API Manager and open the catalog where the OAuth Provider is used.



Configure the catalog to use the resources (2 of 2)

- Enable the Native OAuth Provider from the catalog settings

Manage / Sandbox

Edit OAuth Provider

	TITLE	TYPE
<input checked="" type="checkbox"/>	MainProviderOA	Native

[Cancel](#) [Save](#)

- Enable the user registry for the catalog from the settings

Manage / Sandbox

Edit API User Registries

	TITLE	TYPE	SUMMARY
<input type="checkbox"/>	AuthURLRegistry	Authentication URL	

[Cancel](#) [Save](#)

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-20. Configure the catalog to use the resources (2 of 2)

Enable the Native OAuth Provider and the user registry from the Manage page of the catalog. Products and APIs that are published to this catalog can now use OAuth security.

11.2. Secure an API with OAuth 2.0 authorization

Secure an API with OAuth 2.0 authorization

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-21. Secure an API with OAuth 2.0 authorization

Topics

- Create an OAuth Provider
- ▶ Secure an API with OAuth 2.0 authorization

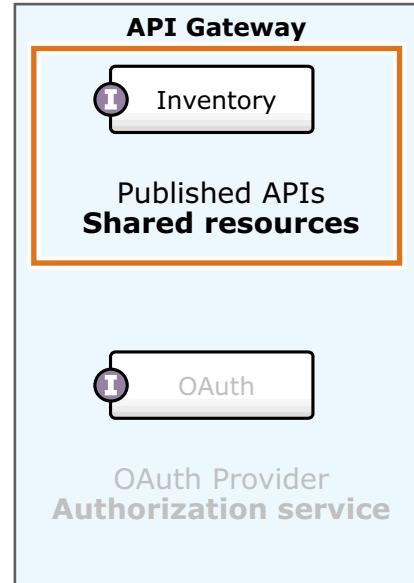
Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-22. Topics

What is an OAuth 2.0 security definition?

- The **OAuth 2.0 security definition** specifies the security settings in the API that you want to secure
- You specify:
 - The name of the **OAuth Provider**
 - Which OAuth 2.0 **message flow** to use
 - The **scope** that is defined by the OAuth Provider



Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-23. What is an OAuth 2.0 security definition?

What is an OAuth 2.0 security definition?

The OAuth 2.0 security definition specifies the security settings in the API that you want to secure.

The following list describes the OAuth 2.0 configuration settings that you can customize in the security definition:

- The OAuth Provider that you want to use
- Which OAuth 2.0 message flow to use
- A scope that matches one of the scopes in the OAuth Provider settings.

Configure OAuth security settings for the API (1 of 3)

- Configure OAuth in the API security definitions



The screenshot shows the IBM API Connect API Manager interface. The left sidebar has icons for Home, API Catalog, API Details, API Setup, Security Definitions (which is selected and highlighted in blue), Security, Paths, and Definitions. The main area shows the API details for 'financing' version 1.0.0. The 'Design' tab is selected. On the right, under 'Security Definitions', there is a table:

NAME	TYPE	LOCATED_IN
clientIdHeader	apiKey	header

A blue 'Add' button is visible next to the table.

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-24. Configure OAuth security settings for the API (1 of 3)

Open the API definition in API Manager. With the Design tab selected, click the Security Definitions option. Click the Add icon.

IBM Training

Configure OAuth security settings for the API (2 of 3)

The screenshot shows the configuration of OAuth security settings for an API. The API is named 'financing' and is at version 1.0.0. The 'Design*' tab is active. The 'Type' section has 'OAuth2' selected. The 'OAuth Provider' dropdown is set to 'MainProviderOA'. The 'Flow' dropdown is set to 'Resource owner password'. The 'Token URL' field contains the URL 'https://\$(catalog.url)/mainprovideroa/oauth2/token'. In the 'Advanced Scope Check' section, there is a table with one row. The row has columns for 'NAME', 'DESCRIPTION', and 'DELETE'. The 'NAME' column contains 'scope1', the 'DESCRIPTION' column contains 'Sample scope description 1', and the 'DELETE' column contains a trash can icon.

NAME	DESCRIPTION	DELETE
scope1	Sample scope description 1	

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-25. Configure OAuth security settings for the API (2 of 3)

Type a name for the OAuth security definition and select the OAuth2 type.

Select the OAuth Provider that you configured earlier and the required flow option.

The token URL field is inserted automatically.

Type the scope that matches any scope that is defined in the OAuth Provider.

Configure OAuth security settings for the API (3 of 3)

- The security definition is added to the list of defined security definitions for the API



NAME	TYPE	LOCATED_IN
OAuth2	oauth2	:
clientIdHeader	apiKey	header

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-26. Configure OAuth security settings for the API (3 of 3)

The named security definition is added to the list of security definitions for the API.



Enable OAuth security for the API

- Select the OAuth security option and the scope to enable OAuth for the API

The screenshot shows the 'Design*' tab selected in the top navigation bar. The main content area is titled 'Security'. It contains a note: 'Security definitions selected here apply across the API, but can be overridden for individual operations. [Learn more](#)'. Below this is a section titled 'SECURITY DEFINITIONS' with two checked items: 'OAuth2 oauth2' and 'clientIdHeader apiKey'. To the right of these is another checked item 'scope1'. At the bottom right of the main area is a blue 'Save' button.

[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-27. Enable OAuth security for the API

Enable OAuth security for the API from the Security option in the Design view.

Click the Add icon.

The OAuth2 security definition that you defined previously can be selected.

When you select the OAuth2 security, the scope option also becomes visible and can be selected.

Save the security settings by clicking the Save icon.

Unit summary

- Explain the concept of an OAuth provider
- Describe the steps to secure an API with OAuth 2.0
- Identify the OAuth Provider types
- Explain how to create a Native OAuth Provider
- Explain the OAuth flow and grant types
- Explain the difference between public and confidential schemes
- Describe how to configure security settings for an API

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-28. Unit summary

Review questions

1. In API Connect, which OAuth grant type setting represents the identity of the client application, rather than the resource owner?
 - A. Implicit
 - B. Password
 - C. Application
 - D. Access code

2. Which of these ways can OAuth scopes be used to define custom access to APIs?
 - A. Included with the authorization request to the user to approve or deny access to the resource.
 - B. Limit access tokens to only the scopes the user has approved.
 - C. Provide different scopes for read or update access to API operations
 - D. All the above.



[Creating an OAuth 2.0 provider](#)

© Copyright IBM Corporation 2020

Figure 11-29. Review questions

Write your answers here:

- 1.

- 2.

Review answers

1. In API Connect, which OAuth grant type setting represents the identity of the client application, instead of the application user?
 - A. Implicit
 - B. Password
 - C. Application
 - D. Access code

The answer is C.

2. Which of these ways can OAuth scopes be used to define custom access to APIs?
 - A. Included with the authorization request to the user to approve or deny access to the resource.
 - B. Limit access tokens to only the scopes the user has approved.
 - C. Provide different scopes for read or update access to API operations.
 - D. All the above. The answer is D.



Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-30. Review answers

Exercise: Declare an OAuth 2.0 Provider and security requirement

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-31. Exercise: Declare an OAuth 2.0 Provider and security requirement

Exercise objectives



- Define a Native OAuth Provider in the API Manager graphical application
- Configure the client ID and client secret security definition
- Declare and enforce an OAuth 2.0 security definition with the API Manager graphical application

Creating an OAuth 2.0 provider

© Copyright IBM Corporation 2020

Figure 11-32. Exercise objectives

Unit 12. Deploying an API to a Docker container

Estimated time

00:45

Overview

This unit examines how to configure and deploy an API implementation to a Docker container. Docker is a lightweight architecture for managing portability and ease of deployment of the application to a runtime environment. You learn how to deploy API implementations to a Docker image and run the API in a Docker container.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Describe what Docker is
- List the benefits of using Docker containers
- Describe the differences between virtual machines and containers
- Provide definitions for some of the terminology that is used with Docker
- Explain how to deploy a LoopBack application to a Docker image

What is Docker?

- Docker is an open source software technology to easily create lightweight, portable, self-sufficient containers from any application
- Build, port, and run any application, anywhere
- Docker is a container platform to efficiently build and manage your applications

Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-2. What is Docker?

Docker is a container platform to efficiently build and manage your applications.

Cargo transport pre-1960

- Challenges of shipping things

Multiplicity
of goods



Multiplicity
of modes of
transport



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-3. Cargo transport pre-1960

The slide depicts some of the challenges of shipping goods before 1960.

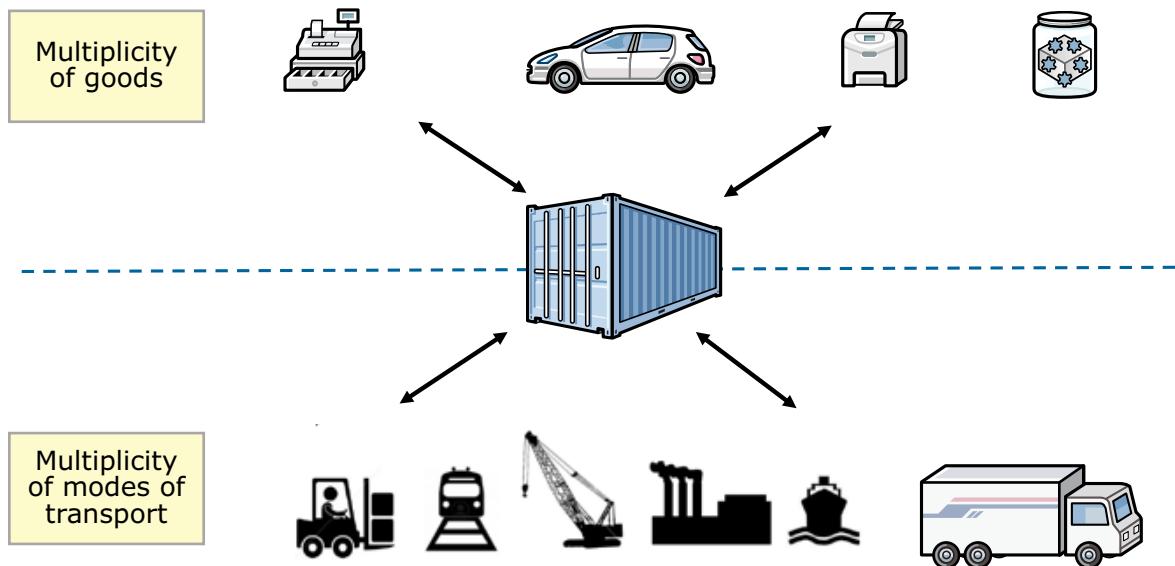
How do you safely and efficiently transport various goods by using different modes of transport?

How do you load and unload the goods from one mode of transport to another?

The answers to these questions are addressed next.

Solution: Cargo container

- Challenges of shipping things



[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

Figure 12-4. Solution: Cargo container

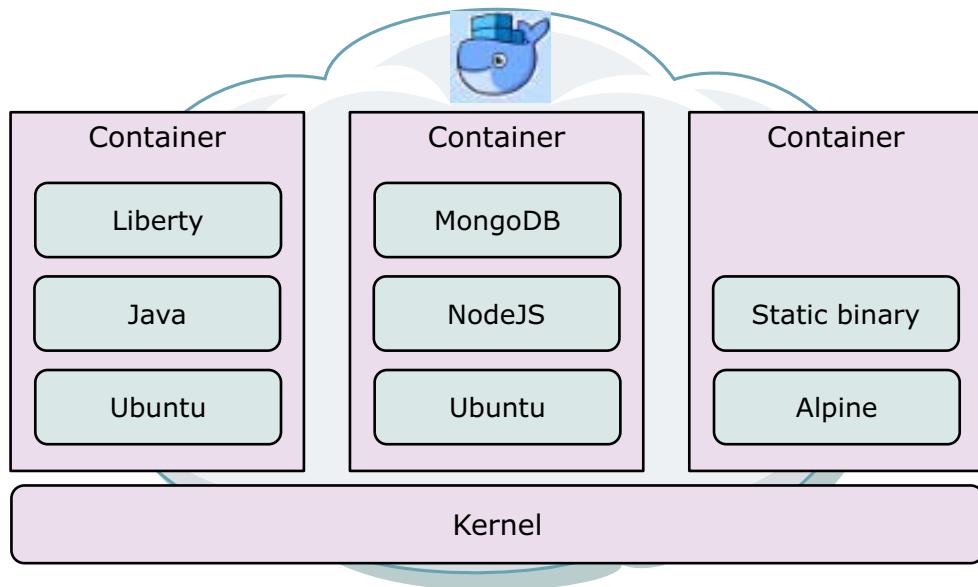
With the cargo container, you can transport a multiplicity of goods.

Goods can be loaded and unloaded, stacked, transported efficiently, and transferred from one mode of transport to another.

To overcome the challenges of a multiplicity of goods that need to be shipped over different modes of transport, containers were introduced. Containers simplify the packaging and distribution of goods in transportation. Using the same analogy, container technology is used to simplify the packaging and deployment of application software for different runtime configurations.

Why Docker containers

- Package software into standardized units for development, distribution, and deployment



[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

Figure 12-5. Why Docker containers

Why use Docker containers?

Containers isolate software from its surroundings. Docker containers can be used to isolate development and staging environments and help reduce conflicts between teams that are running different software versions on the same infrastructure. For example, development and staging environments might use a different version of a Java runtime environment.

Why use Docker containers?

- Benefits for developers
 - A clean, safe, and portable runtime environment
 - No worries about missing dependencies, packages, and other items
 - You build the application once
 - Run each application in its own isolated container, so you can run various versions of libraries and other dependencies
 - Automation of testing, integration, and packaging
 - Reduces or eliminates concerns about compatibility on different platforms
- Benefits for operations
 - Makes the entire lifecycle more efficient, consistent, and repeatable
 - Increases the quality of code that developers produce
 - Eliminates inconsistencies between development, test, and production environments
 - Supports segregation of duties
 - Significantly improves the speed and reliability of continuous deployment and integration

[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

Figure 12-6. Why use Docker containers?

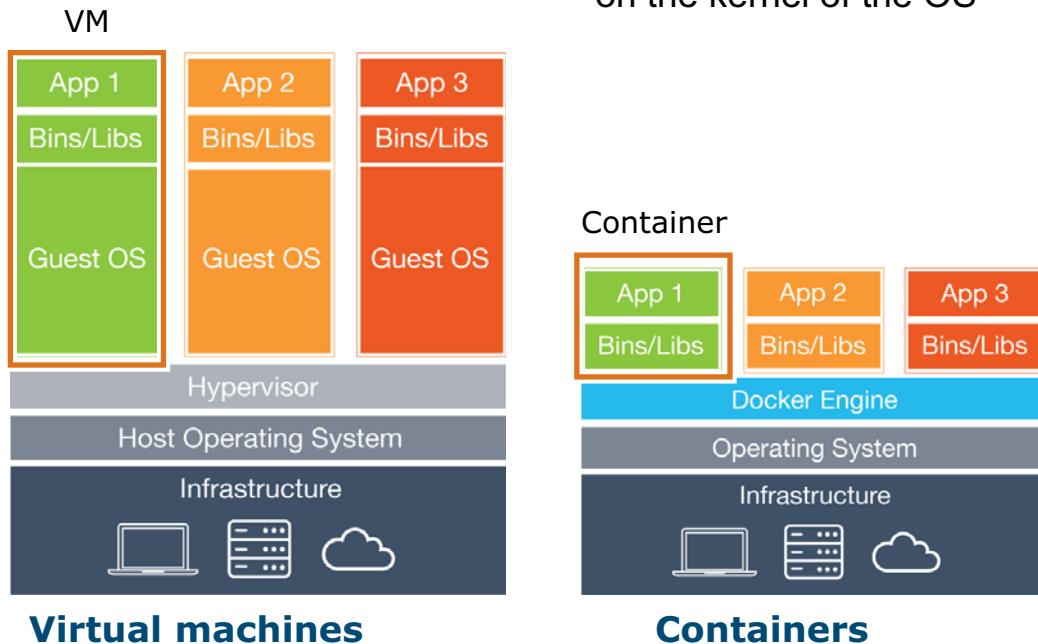
Using containers to run applications is beneficial in many ways.

Developers can avoid dependency conflicts by specifying exact versions of the software at build time. Each application can be run in its own isolated container so that developers can run various versions of libraries and other dependencies. Developers can automate the deployment, integration, packaging, and testing of applications that run on containers.

Operations staff members benefit from containerization by making the application lifecycle more efficient, consistent, and repeatable. The use of containers significantly improves the speed and reliability of continuous deployment and integration.

Virtual machines versus containers

- Virtual machines run guest OSs
- Containers run the apps natively on the kernel of the OS



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-7. Virtual machines versus containers

Virtualization is a technology in which an application, guest operating system, or installed software is abstracted away from the true underlying hardware and infrastructure. Virtualization uses a software layer that is called a hypervisor to emulate the underlying hardware.

Virtual machines, or VMs, differ from containers in that they include an entire guest operating system that can be different from the host operating system.

Docker provides another layer of abstraction of operating-system-level virtualization on Windows and Linux.

Definition: Hypervisor

- A hypervisor is a function that abstracts – isolates – operating systems and applications from the underlying computer hardware
- The abstraction allows the underlying host hardware to independently operate one or more virtual machines as guests
 - In this manner, multiple guest VMs can effectively share the system's physical compute resources, such as processor cycles, memory space, and network bandwidth
- A hypervisor is sometimes also called a virtual machine monitor

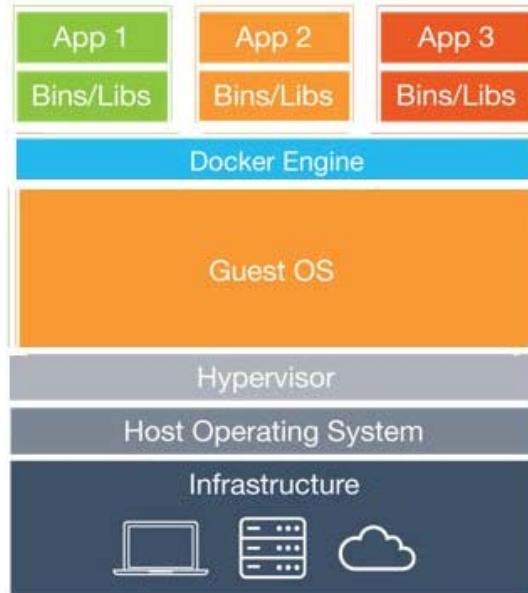
Figure 12-8. Definition: Hypervisor

Virtualization technology uses a software layer that is called a hypervisor to emulate the underlying hardware.

Containers are lightweight because they do not need the extra load of a hypervisor, but run directly within the kernel of the host machine.

Containers on virtual machines

- Docker is supported on top of guest operating systems
 - You can even run Docker containers within host systems that are virtual machines



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-9. Containers on virtual machines

In the Docker documentation, see:

<https://docs.docker.com/engine/docker-overview/#the-docker-platform>

Docker containers are supported on top of guest operating systems.

In the lab exercises for this course, you run Docker containers on top of the student guest virtual machine that runs the Ubuntu operating system.

Docker terminology

- **Image**
 - Executable package that includes everything that is needed to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files
 - Layered file system where each layer references the layer beneath
- **Dockerfile** – build script that defines:
 - Uses an existing image as the starting point
 - A set of instructions to augment that image (each of which results in a new layer in the file system)
 - Metadata such as the ports that are exposed
 - The command to execute when the image is run
- **Container**
 - Runtime instance of an image plus a read/write layer
 - The container runs isolated from the host environment by default, accessing host files and ports only if configured to do so
- **Docker Hub**
 - Centralized repository of Docker images

[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

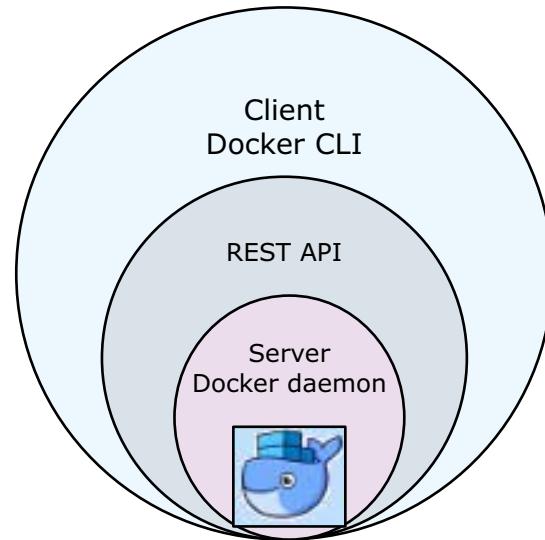
Figure 12-10. Docker terminology

This page introduces some of the terminology that is used with Docker containers:

- **Image:** Layered file system where each layer references the preceding layer.
- **Dockerfile:** Contains the build script to create the Docker image. The Dockerfile is a text file that contains all the commands that are run in sequence to build the Docker image. The build script defines:
 - An existing image as the starting point.
 - A set of instructions to augment that image (each of which results in a new layer in the file system).
 - Metadata such as the ports that are exposed.
 - The command to execute when the image is run.
 - When you type the `docker run` command, a runtime instance of an image is created as a Docker container. The container image includes everything that is needed to run the application: code, runtime, system tools, system libraries, and settings.
- **Container:** Runtime instance of an image plus a read/write layer.
- **Docker Hub:** Centralized public repository of Docker images.

Docker Engine

- Docker Engine is a client/server application with these major components:
 - A server, which is a long-running process that is called a daemon process (the `dockerd` command)
 - A REST API that specifies how programs can communicate with the daemon and instruct it on what to do
 - A command line interface (CLI) client (the `docker` command)



[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

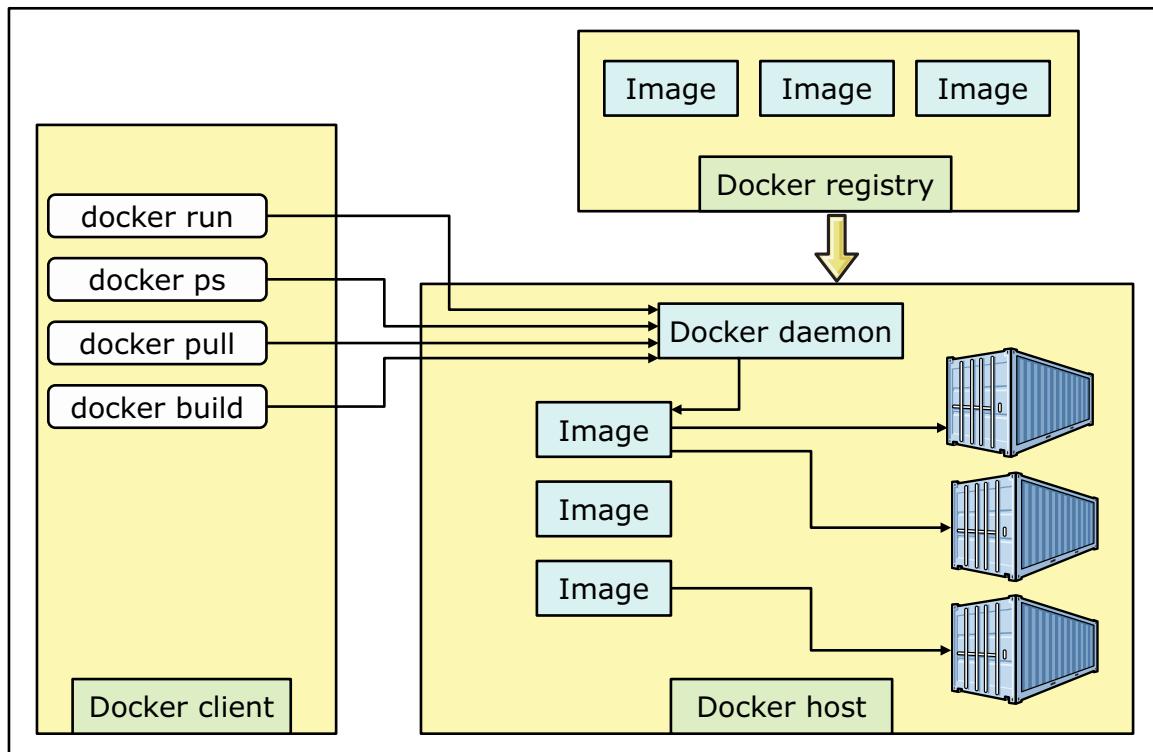
Figure 12-11. Docker Engine

You can view the commands from the CLI by typing: `dockerd --help` and `docker --help`

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client (`docker`) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out. The `docker` command uses the Docker API.

Docker architecture



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-12. Docker architecture

This diagram shows the Docker architecture.

The Docker client (the `docker` command) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to the Docker daemon, which carries out the command. The `docker` command uses the Docker API.

Most of the images that you create are built on top of a base image from the Docker Hub. Docker Hub contains many prebuilt images that you can download and try without the need to define and configure your own. To download a particular image from the Docker Hub, use the `docker pull` command. The image is downloaded and installed on the Docker host.

Docker version

- The Docker version shows the version of Docker and the Docker API

```
localuser@ubuntu-vm:~$ docker version
Client:
Version:          18.09.1
API version:      1.39
Go version:       go1.10.6
Git commit:       4c52b90
Built:            Wed Jan  9 19:35:23 2019
OS/Arch:          linux/amd64
Experimental:     false

Server: Docker Engine - Community Engine:
Version:          18.09.1
API version:      1.39 (minimum version 1.12)
Go version:       go1.10.6
Git commit:       4c52b90
Built:            Wed Jan  9 19:02:44 2019
OS/Arch:          linux/amd64
Experimental:     false
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-13. Docker version

The output from running the `docker version` CLI command is displayed. The output shows which versions of the Docker client and server are installed.

Docker CLI

- Example `docker run` command

```
localuser@ubuntu:~$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be
working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the
    Docker Hub.
 3. The Docker daemon created a new container from that image
    that runs the
      executable that produces the output you are currently
      reading.
 4. The Docker daemon streamed that output to the Docker
    client, which sent it
      to your terminal.
...
localuser@ubuntu:~$
```

Figure 12-14. Docker CLI

This page shows the output from running the `docker run hello-world` CLI command. The container with the hello-world application is started. The hello-world container is often used to verify the installation of the Docker software.

Docker registry

- A registry is a storage and content delivery system, holding named Docker images, available in different tagged versions
 - A registry is a library of images
- Docker Hub
 - An instance of a public registry
 - A Cloud hosted service from Docker
 - Provides registry capabilities for public and private content
 - Responsible for centralizing information about user accounts, images, and public name spaces
- Docker hub hosts official repositories from numerous vendors
- A common starting point is to download content from the Docker hub with the operating system or middleware you intend to use as a basis for your container

[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

Figure 12-15. Docker registry

The Docker registry is a storage and content delivery system, holding named Docker images, available in different tagged versions.

Docker Hub is:

- An instance of a public registry
- A Cloud-hosted service from Docker
- A provider of registry capabilities for public and private content
- Responsible for centralizing information about user accounts, images, and public name spaces.

Docker Hub hosts official repositories from a number of vendors.

A common starting point is to download content from the Docker hub with the operating system or middleware that you intend to use as a basis for your container.

Docker Hub

- Examples of docker images available
 - Base Ubuntu
 - WebSphere Liberty
 - MongoDB
- Most images require other existing images and build on them
- Use the `docker pull` command to download images from Docker Hub registry
- You can create a private registry
 - `docker pull registry`
- Run a local registry
 - `docker run -d -p 5000:5000 --restart always --name registry registry:2`

Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-16. Docker Hub

Docker Hub contains both operating system base images and pre-built images that you can download and try without needing to define and configure your own image.

Most images require other existing images and build on them.

For example, the WebSphere Liberty image might build on the base Ubuntu image.

In addition to the Docker Hub public registry, you can create your own private registry and repository for your images.

Use the command `docker pull registry` to install a private registry. Then, use the `docker run` command to start the registry.

Dockerfile

- A simple, descriptive set of steps that are called instructions, which are stored in a Dockerfile
 - Each instruction creates a layer in the image
- You create or extend the Dockerfile to define your image and use a build command to create the image
- Dockerfile
 - A build script that defines an image as the starting point
 - Tells the image builder what the image contains
 - Set of instructions to augment that image
 - Defines any metadata

```
FROM websphere-liberty
ADD app.war /opt/ibm/wlp/usr/servers/defaultServer/dropins/
ENV LICENSE accept
```

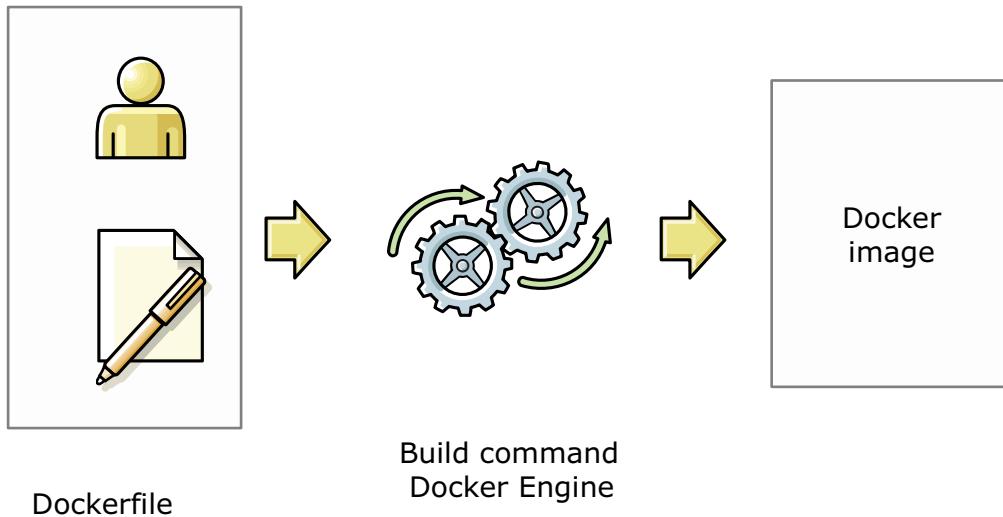
Figure 12-17. Dockerfile

Dockerfile is a build script that defines:

- An existing image as the starting point
- A set of instructions to augment that image, each of which results in a new layer in the file system
- Metadata such as the ports that are exposed
- Commands to start any processes inside the image

Building images from a Dockerfile

- The Docker build command runs the build script and creates a layer in the file system for each instruction and saving the resultant image in a local registry



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-18. Building images from a Dockerfile

The docker build command executes the build script and creates a layer in the file system for each instruction. The resultant image is saved on the local file system or in a local registry. A Docker image is created when the build command runs successfully.

Dockerfile example: Inventory application

```
# Build with: docker build -t apic-inventory-image .
# Create a small alpine Linux distribution as the base image
FROM alpine:3.8
RUN apk add --update nodejs nodejs-npm && npm install npm@5.6.0 -g
WORKDIR /inventory
# Copy the files in the host current dir to the Docker WORKDIR
ADD . /inventory
RUN pwd;ls
# Make port available outside the container
EXPOSE 3000
#CMD to start
CMD ["npm","start"]
# Run the image with the command:
# docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-19. Dockerfile example: Inventory application

Dockerfile is a text file that defines the contents and startup behavior of a single container.

The image is built on a small alpine Linux distribution as the base image. The script adds the nodejs and npm software to the image, then creates a work directory on the image. Next, the build script copies the files in the current host directory to the image work directory, and then displays the contents of the work directory. Next, the build script makes the port 3000 available outside the container. Finally, the build script runs the command to start the application on the image.

The image is built by using the command `docker build --tag` followed by the image name and the path. In this way, all the files in the path directory are sent to the Docker daemon in the build step.

The page shows the Dockerfile source code for building the apic-inventory-image for the inventory application that is used in the course exercises. The following keywords are important:

- **FROM** – The build uses an alpine base image.
- **RUN** – Install nodejs, nodejs-npm, and npm on the image.
- **WORKDIR** – Create a directory to hold the application inside the image. A directory is created on the image.
- **ADD** – The files are copied from the host current directory to the directory on the image.
- **RUN** – Displays the directory and files on the image.

- **EXPOSE** – Port 3000 is exposed for access from outside the container.
- **CMD** – The npm start command is issued in the image.

Build the image with the command:

```
docker build -t <image-name> <path>
```

The example specifies that the build uses the `-tag` or `-t` option, which gives a name and optionally a tag in the `name:tag` format to the image.

The example specifies that the path is `(.)` and so all the files in the local directory are sent to the Docker daemon.

Listing images

- Use the command `docker images` to display the list of images that are installed

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
apic-inventory-image	latest	e7ad6f015d63	2 weeks ago	125MB
alpine	3.8	dac705114996	2 weeks ago	4.41MB
quay.io/calico/node	v3.5.0	0071925dc13a	2 months ago	71.7MB
quay.io/calico/cni	v3.5.0	582b0cf3f11a	2 months ago	83.6MB
quay.io/calico/kube-controllers	v3.5.0	435d6b2acd79	2 months ago	50.5MB
registry	2	116995fd6624	2 months ago	25.8MB

[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

Figure 12-20. Listing images

The `docker images` command shows all top-level images, their repository and tags, and their size. The SIZE is the cumulative space that the image and all its parent images take.

Run the application on the Docker container

- Usage

- `docker run [options] image [command] [arg...]`

```
docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image

> inventory@1.0.0 start /inventory
> node .

Web server listening at: http://localhost:3000
Browse your REST API at: http://localhost:3000/explorer
```

Figure 12-21. Run the application on the Docker container

Use the `docker run` command to run the application on the Docker container. The Docker container is the runtime instance of the image.

- The `--add-host` option adds custom host-to-IP mappings.
- The `--publish` or `-p` option adds a container's port to the host.

For example, the command

`$ docker run -p 127.0.0.1:80:8080 ubuntu bash`

binds port 8080 of the container to port 80 on 127.0.0.1 of the host machine.

Listing containers

- docker ps shows all running containers

```
localuser@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
REATED
057bc1155d34      apic-inventory-image   "npm start"        3 minutes ago     Up 3 minutes      0.0.0.0:3000->3000/tcp   practical_gauss
f3c3237fa169      b629c847c5ef          "/bin/node /app/app..."  5 hours ago       Up 5 hours        k8s_apim_r674f0b
c86d-apim-v2-88674dd9f-g8r95_apiconnect_77ade2d7-24da-11e9-9455-00505682a85e_20
d1be5e8a7daa      90556f9b55bf          "/bin/node /app/app..."  5 hours ago       Up 5 hours        k8s_lur_r674f0bc

```

- docker ps -a shows all containers, including containers that are not running

```
File Edit View Terminal Tabs Help
student@xubuntu-vm:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
STATUS
499659843f13      apic-inventory-image   "npm start"        4 days ago       Exited (0) 4 days ago
47a4c3206d92      apic-inventory-image   "npm start"        5 days ago       Exited (0) 4 days ago
student@xubuntu-vm:~$
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-22. Listing containers

The docker ps command can be used to inspect the state of containers.

The docker ps command shows just the running containers.

The command docker ps with the --all or -a option shows all containers, including the stopped containers.

Useful Docker commands

- Run in the background with a port that is exposed on the host

```
docker run --add-host mysql.think.ibm:192.168.225.10 --add-host
mongo.think.ibm:192.168.225.10 -p 3000:3000 -detach -name inv
apic-inventory-image

docker logs -tail=all -f inv
```

- Allow Docker to allocate ports when multiple containers are running

```
C=$(docker run -e LICENSE=accept -P -d websphere-liberty)
docker port $C 9080
docker logs -tail=all -f $C
```

- And when you are done

```
docker stop $C
docker rm $C
```

- Remove all Docker containers (force)

```
docker rm -f $(docker ps -a -q)
```

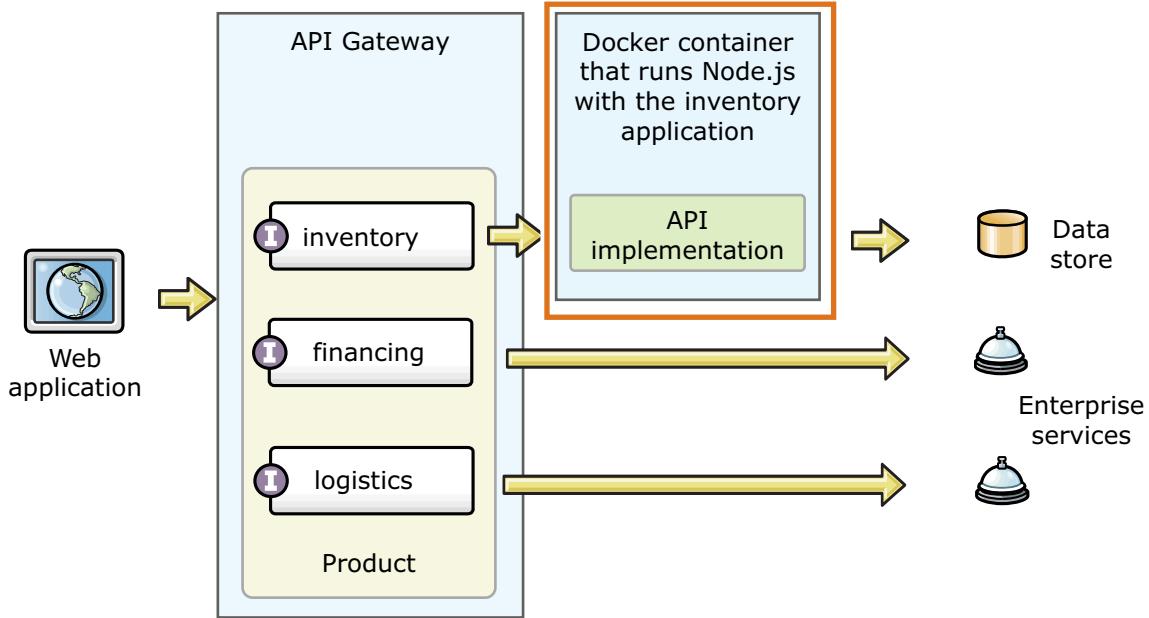
Figure 12-23. Useful Docker commands

To get a list of Docker commands from the terminal emulator, type: `docker --help`

Run `docker <command> --help` for more information on each command.

For a complete list of the docker CLI commands, see the Reference section of the online Docker documentation at docs.docker.com. You can also type `docker -help` from the terminal of a machine where Docker is installed.

API runtime message flow: exercise case study



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-24. API runtime message flow: exercise case study

In the exercise case study, you created three API definitions in API Manager that are called on the API Gateway. You also created the inventory LoopBack application, which is an implementation of an API that runs as a Node application on the workstation.

The next step is to implement the LoopBack application as a Node application that runs as a Docker container. When you run the application as a Docker container, you make the application more portable and scalable.

The product is a collection of APIs that are organized together. You can publish many APIs in a product.

Unit summary

- Describe what Docker is
- List the benefits of using Docker containers
- Describe the differences between virtual machines and containers
- Provide definitions for some of the terminology that is used with Docker
- Explain how to deploy a LoopBack application to a Docker image

[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2020

Figure 12-25. Unit summary

Review questions

1. True or False: When you publish LoopBack applications to API Manager, the API Manager hosts and runs the application.
2. Which of these commands creates a Docker image?
 - A. docker images
 - B. docker run
 - C. docker build
 - D. docker rm



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-26. Review questions

Write your answers here:

- 1.
- 2.

Review answers

1. True or False: When you publish LoopBack applications to API Manager, the API Manager hosts and runs the application.

The answer is False. The API Manager does not host the LoopBack application: the Docker container instance runs the application.



2. Which of these commands creates a Docker image?

- A. docker images
- B. docker run
- C. docker build
- D. docker rm

The answer is C.

Exercise: Deploy an API implementation to a container runtime

Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-28. Exercise: Deploy an API implementation to a container runtime

Exercise objectives

- Test a local copy of a LoopBack API application
- Examine the Dockerfile that is used to deploy a Loopback API
- Create a Docker image by using the docker build command with the Dockerfile
- Verify that the API runs on the Docker image



Deploying an API to a Docker container

© Copyright IBM Corporation 2020

Figure 12-29. Exercise objectives

Unit 13. Publishing and managing products and APIs

Estimated time

01:00

Overview

This unit examines how to package and publish APIs to the API Connect cloud. A product defines a collection of APIs for deployment. The product contains a plan, which is a contract between the API provider and API consumer that specifies quality of service characteristics, such as the rate limit of API calls.

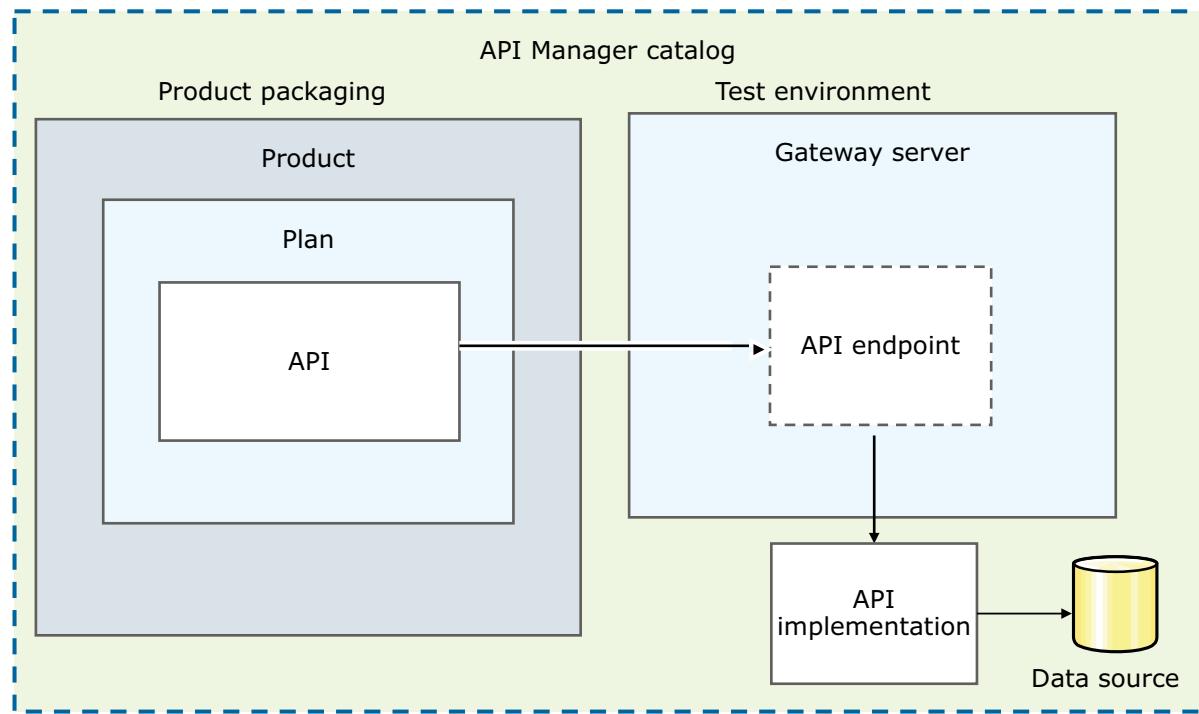
How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Explain the concept of a plan, a product, and a catalog
- Explain the staging and publishing API lifecycle stages
- Define an API product and a plan
- Describe the steps to publish a product
- Describe the Developer Toolkit interface for managing products and APIs
- Explain the lifecycle states for products and APIs

Product, plan, API hierarchy



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-2. Product, plan, API hierarchy

Products, plans, and APIs are all represented in YAML files within API Manager.

Testing the API operations:

You can test your API endpoint operations from the API Manager assembly test feature or from the Developer Portal for a catalog. The API and product must either be auto-published by the assembly test feature or explicitly published to the catalog. Publishing the API and product makes the API callable on the Gateway server. The application implementation must be started and reachable from the Gateway.

Select the API REST operation that you want to test. Then, call the operation with the test client that runs on API Manager or the API Developer Portal.

Define product and plan

- *Products* provide a method by which you can group APIs into a package
 - Can contain plans
 - Add API operations to the product
 - Products are published to a *catalog*
- *Plan*: To make an API available to an application developer, it must be included in a plan
 - Can be used to enforce rate limits
 - Can be used for billing by separating plans into free or charge or billable plans

Figure 13-3. Define product and plan

You can create plans only within products, and these products are then published in a catalog.

Multiple plans within a single product are useful in that they can fulfill similar purposes but with differing rate limits or cost structures.

API product definition file

```

info:
  version: 1.0.0
  title: hello-world
  name: hello-world
gateways:
  - datapower-api-gateway
plans: default-plan:
  rate-limits:
    default:
      value: 100/1hour
    title: Default Plan
    description: Default Plan
    approval: false
apis:
  helloworld1.0.0:
    name: 'helloworld:1.0.0'
visibility:
  view:
    type: public

```

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-4. API product definition file

The API product definition file uses the same file structure as an OpenAPI definition. API products are specific to the IBM API Connect product and are extensions to the OpenAPI specification.

The product definition file can be viewed from the Design view or the Source view in API Manager.

Example

In the student environment, you can generate the helloworld application with the command **apic lb**. Then, select the helloworld example.

Create the OpenAPI definition with the command **apic lb export-api-def > helloworld.yaml**.

Import the YAML file into API Manager. Create the hello-world product in API Manager and select the helloworld API to add to the product. The product YAML file can be view from the source view in API Manager when you edit the product.



Add a product (1 of 3)

The screenshot illustrates the 'Add Product' process in the IBM API Connect API Manager. It is divided into three main sections:

- Left Panel (1):** Shows the 'Develop' section with the 'APIs and Products' list. An 'Add' icon is highlighted, and a yellow circle with the number '1' is placed over it.
- Middle Panel (2):** The 'Add Product' dialog box. Under the 'Create' tab, 'New product' is selected. A yellow circle with the number '2' is placed over the 'Next' button.
- Right Panel (3):** The 'Create New Product' dialog box. It contains fields for 'Info' (Title: hello-world, Name: hello-world, Version: 1.0.0), and a yellow circle with the number '3' is placed over the title field.

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-5. Add a product (1 of 3)

To add a product in API Manager, open the Develop APIs and Products page.

1. From the APIs and Products page, click the **Add** icon. Then, select **Product**.
2. Select **New product**. Click **Next**.
3. Type a title, name, and version for the product. Click **Next**.
4. Select the APIs that you want to add to the product. Click **Next**.
5. Select the plan for the product. Click **Next**.
6. Select the option whether to publish the product. Select the visibility and subscribability options. Click **Next**.
7. The Summary page is displayed.
8. Edit the product in the Design view to review or change the product definition.

IBM Training

Add a product (2 of 3)

Create New Product

APIs
Select the APIs to add to this product

<input type="checkbox"/> TITLE	VERSION	DESCRIPTION
<input type="checkbox"/> financing	1.0.0	
<input checked="" type="checkbox"/> helloworld	1.0.0	
<input type="checkbox"/> inventory	1.0.0	
<input type="checkbox"/> ItemService	1.0.0	
<input type="checkbox"/> logistics	1.0.0	
<input type="checkbox"/> notes	1.0.0	
<input type="checkbox"/> savings	1.0.0	

Back **Cancel** **Next**

Create New Product

Plans
Add plans to this product **Add**

Default Plan **5**

Title
Default Plan

Description (optional)
Default Plan

Rate Limit
100 / 1 hour

Back **Cancel** **Next**

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-6. Add a product (2 of 3)

To add a product in API Manager, open the Develop APIs and Products page.

1. From the APIs and Products page, click the **Add** icon. Then, select **Product**.
2. Select New product. Click **Next**.
3. Type a title, name, and version for the product. Click **Next**.
4. Select the APIs that you want to add to the product. Click **Next**.
5. Select the plan for the product. Click **Next**.
6. Select the option whether to publish the product. Select the visibility and subscribability options. Click **Next**.
7. The Summary page is displayed.
8. Edit the product in the Design view to review or change the product definition.

IBM Training

Add a product (3 of 3)

The screenshot shows two panels of a product creation wizard:

- Left Panel (Step 6): Visibility**
 - Select the organizations or groups you would like to make this product visible to.
 - Options: Public, Authenticated, Custom.
- Right Panel (Step 7): Summary**
 - Created new product
 - Added APIs
 - Added rate limits

Both panels have a yellow circle with the number corresponding to the step (6 or 7) in the top right corner.

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

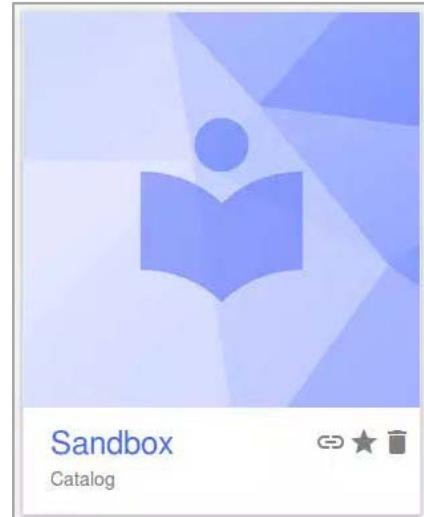
Figure 13-7. Add a product (3 of 3)

To add a product in API Manager, open the Develop APIs and Products page.

- From the APIs and Products page, click the **Add** icon. Then, select **Product**.
- Select New product. Click **Next**.
- Type a title, name, and version for the product. Click **Next**.
- Select the APIs that you want to add to the product. Click **Next**.
- Select the plan for the product. Click **Next**.
- Select the option whether to publish the product. Select the visibility and subscribability options. Click **Next**.
- The Summary page is displayed.
- Edit the product in the Design view to review or change the product definition.

Catalogs

- Catalogs are useful for separating products and APIs for testing and production
- The URLs for API calls and the Developer Portal are specific to a particular catalog
- By default, a Sandbox catalog is provided
 - Used for testing
- Organization owners create more catalogs
 - Production catalog for hosting APIs that are ready for use



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-8. Catalogs

Products must be staged and published to a catalog to become available to application developers.

In a typical configuration, an API provider organization uses a Sandbox catalog for testing APIs under development and a production catalog for hosting APIs that are ready for full use.

Catalogs are usually configured and managed from the API Manager user interface.

Stage a product and API

- Makes the product and associated APIs visible to the catalog
- From within the catalog, the product can be managed through the lifecycle states

APIs and Products			Add ▾
TITLE	TYPE	LAST MODIFIED	
 financing-1.0.0	API (REST)	21 days ago	...
 helloworld-1.0.0	API (REST)	a day ago	...
 inventory-1.0.0	API (REST)	21 days ago	Publish
 ItemService-1.0.0	API (WSDL)	22 days ago	Stage

- Prompts whether to publish a new product or an existing product

Stage API

<input checked="" type="radio"/> New Product Publish using a new product	<input type="radio"/> Existing Product Publish using an existing product
---	---

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-9. Stage a product and API

The stage action creates a snapshot of the product and its associated APIs and makes them visible on the catalog.

Stage is the first lifecycle state that is managed from within the catalog.

When in the staged state, the product is not yet visible or subscribable to developers on the Developer Portal.



Publish a product and API (1 of 2)

Publish from the API Manager

- Click the options icon in the row with the product that you want to publish. Then, click **Publish**

Develop		
savings-1.0.0	API (REST)	23 days ago
financing auto product-1.0.0	Product	21 days ago
hello-world-1.0.0	Product	20 hours ago
inventory auto product-1.0.0	Product	21 days ago
ItemService auto product-1.0.0	Product	22 days ago

...

Publish (1)

Stage

Save as a new version

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-10. Publish a product and API (1 of 2)

You publish the product and its associated APIs from the list of APIs and Products page in the Develop option of API Manager.

Click the list of options ellipsis in the product row. Then, select Publish.

You can publish a product without first going through the stage lifecycle state.

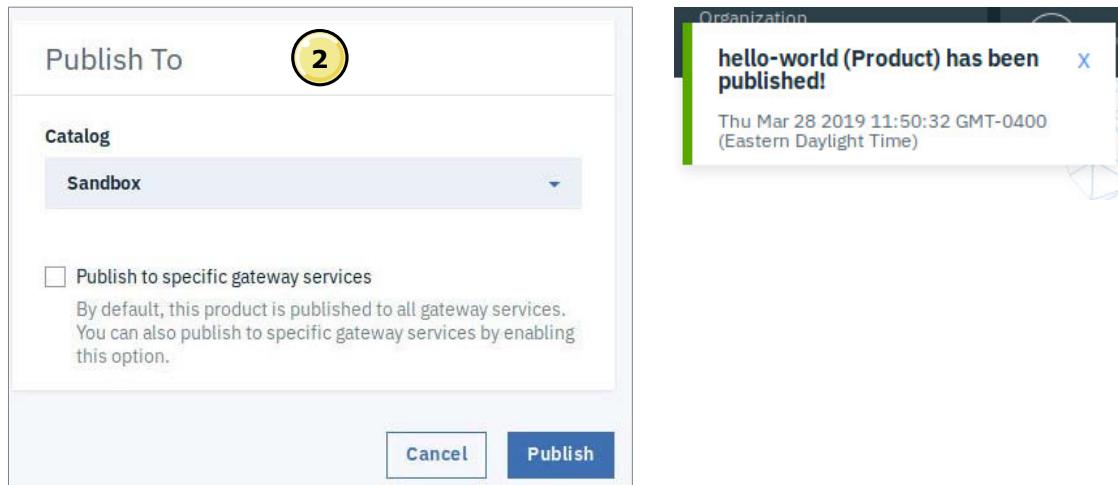
Publishing a product makes it visible on the Developer Portal. The API also becomes callable on the API gateway.



Publish a product and API (2 of 2)

Publish from the API Manager

- Select the target catalog for publishing
- Optionally, choose a specific gateway service
- Click **Publish**



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-11. Publish a product and API (2 of 2)

You publish the product and its associated APIs from the list of APIs and Products page in the Develop option of API Manager.

On the publish product page, select the target catalog from the drop-down list.

Optionally select to publish to a specific gateway service.

Click Publish.

The product is published.



Review the published products (1 of 2)

- From the Manage option in API Manager, select the target publishing catalog, for example, Sandbox

TITLE	NAME	STATE	
> financing auto product	financing-auto-product 1.0.0	Published	⋮
> hello-world	hello-world 1.0.0	Published	⋮
> inventory auto product	inventory-auto-product 1.0.0	Published	⋮
> ItemService auto product	itemservice-auto-product 1.0.0	Published	⋮
> logistics auto product	logistics-auto-product 1.0.0	Published	⋮
> savings auto product	savings-auto-product 1.0.0	Published	⋮
> think-product	think-product 1.0.0	Published	⋮

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-12. Review the published products (1 of 2)

From the Manage option in API Manager, select the target publishing catalog, for example, Sandbox.

The list of published products is displayed on the products page.



Review the published products (2 of 2)

- Expand the published product to display the associated APIs, plans, and gateway type

TITLE	NAME	STATE	
> financing auto product	financing-auto-product 1.0.0	Published	⋮
hello-world	hello-world 1.0.0	Published	⋮
APIs		STATE	
helloworld:1.0.0		online	⋮
PLANS			
Default Plan			
GATEWAY SERVICES		GATEWAY TYPE	
DataPower API Gateway		DataPower API Gateway	

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-13. Review the published products (2 of 2)

From the Manage option in API Manager, select the target publishing catalog, for example, Sandbox.

The list of published products is displayed on the products page.

Expand the published product to display the associated APIs, plans, and gateway type.

Review from the terminal: Toolkit interface (1 of 2)

- Sign in to API Manager

```
$ apic login
Enter your API Connect credentials
Server? manager.think.ibm
Realm? provider/default-idp-2
Username? thinkowner
Password?
Logged into manager.think.ibm successfully
```

- To determine the realm value to use when you sign on, type the command:

```
$ apic identity-providers:list --scope provider -s
manager.think.ibm
default-idp-2
```

Figure 13-14. Review from the terminal: Toolkit interface (1 of 2)

Actions that are done in the API Manager graphical application have equivalent API Connect Developer Toolkit commands. These commands are typed in the terminal command line interface.

The interactive apic login command is displayed. The realm value depends on the user registry that is used by API Manager. When a local user registry is used in API Manager, the realm value is provider/default-idp-2. If you use an LDAP user registry, the value is provider/ldap.

Review from the terminal: Toolkit interface (2 of 2)

- List catalogs

```
$ apic catalogs:list -o think -s manager.think.ibm  
sandbox https://manager.think.ibm/api/catalogs/7ca...
```

- List the status of the hello-world product in the sandbox catalog:

```
$ apic products:list -c sandbox -o think -s manager.think.ibm  
--scope catalog hello-world  
hello-world:1.0.0 {state: published}  
https://manager.think.ibm/api/catalogs/7ca...
```

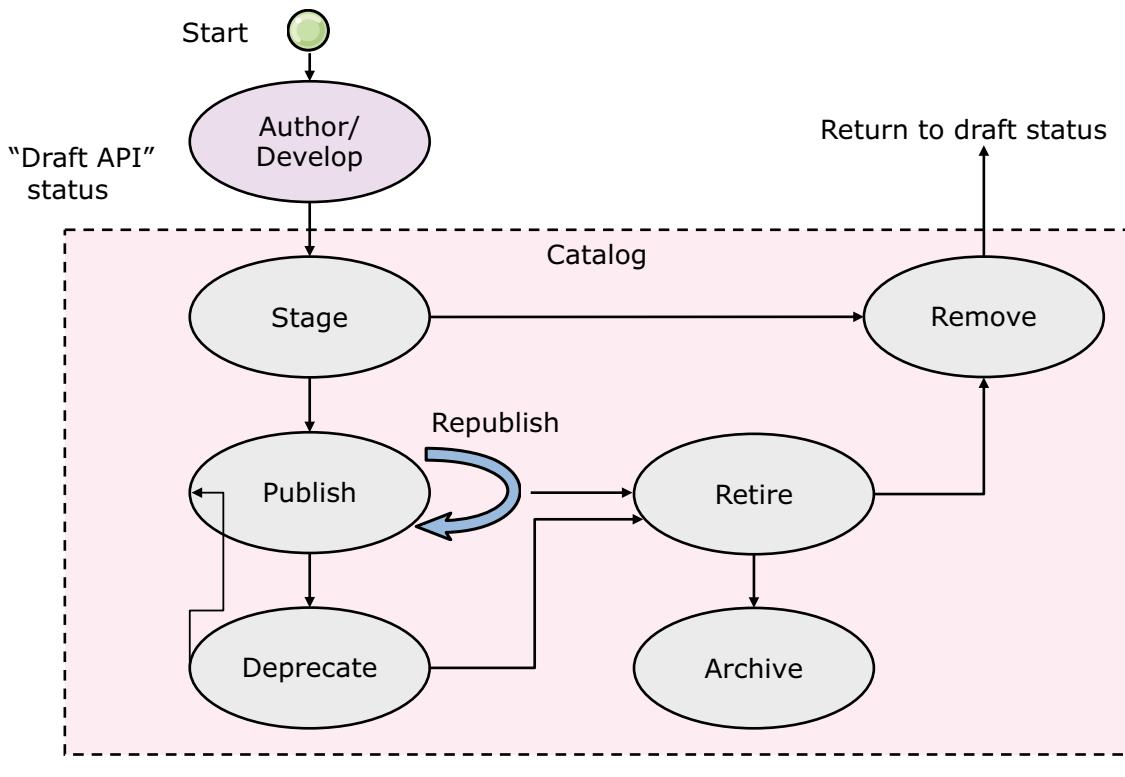
Figure 13-15. Review from the terminal: Toolkit interface (2 of 2)

Actions that are done in the API Manager graphical application have equivalent API Connect Developer Toolkit commands. These commands are typed in the terminal command line interface.

The interactive apic login command is displayed. The realm value depends on the user registry that is used by API Manager. When a local user registry is used in API Manager, the realm value is provider/default-idp-2. If you use an LDAP user registry, the value is provider/ldap.

When you are signed on to API Manager from the terminal, you can type commands that query the status of products and APIs in the catalog.

Lifecycle of products and API resources



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-16. Lifecycle of products and API resources

Here you see a lifecycle for products and their contained plans and API operations as they move through the different states.

The whole lifecycle of products, plans, and APIs occurs in the context of a catalog.

Except for the authoring step, all the actions involve state changes to products, plans, and API operations within a particular catalog.

When you first create the API in the draft API status, the API and its associated product exist independently of the catalog. The Develop page in API Manager keeps all the APIs and products in the “draft” status of the figure.

The next step is that you stage the product and its contained API resources to the catalog.

You then publish the product to make it visible on the Developer Portal for that catalog.

When a product is moved to the deprecated state, the plans in the product are visible only to developers whose applications are currently subscribed. No new subscriptions to the plan are possible.

A product version in the retired state cannot be viewed or subscribed to, and all of the associated APIs are stopped.

Product versions in the archived state are similar to ones in the retired state. However, archived product versions are not displayed by default on the products view of the API Manager.

Stage and publish a previously published product (1 of 2)

- Target catalog: Sandbox
 - Publishing a product from API Manager to a Sandbox catalog where the product is already published results in publish of the **same version** of the product
 - Republish a product from the list of APIs and products from the Develop page in API Manager
 - When you publish from the Assembly view test feature, the product overwrites the existing published auto-product that gets created automatically by the test feature

Figure 13-17. Stage and publish a previously published product (1 of 2)

If you publish a product to a Sandbox catalog and change it in API Manager, you can publish the same version of the product. Publishing the same version to the Sandbox catalog overwrites the existing published product with the newer published version.

When you work with APIs and products on the Sandbox catalog, API Connect assumes that you are working in development mode. The changes that you make to these artifacts overwrite the earlier versions when you publish them to the Sandbox catalog.

Stage and publish a previously published product (2 of 2)

- Target catalog: non-sandbox
 - A published product in a non-sandbox catalog exists independently from the product that you edit or publish in the API Manager
 - Publishing a product to a non-sandbox catalog where the product is already published results in another instance of the product version
 - For this reason, it is advised that when you stage a product, you then **create a new version** of the product in the API Manager to make future updates

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-18. Stage and publish a previously published product (2 of 2)

If you publish a product to a non-sandbox catalog, the product that is published is an independent and fixed copy of any subsequent published version.

Before you stage or publish a product to a non-development catalog, it is suggested that you then create a version of the product. The already-published product is at the earlier version number. You can then edit and change the later product version and APIs. API Connect is not a version control tool. You manually change the version number strings or use the create version option in API Manager.

API Manager prompts you to create a new version of a product when you try to publish an existing published product version.

Manage published products in API Manager

- Published products can be viewed and managed in API Manager

TITLE	NAME	STATE
> financing auto product	financing-auto-product 1.0.0	Published
> hello-world	hello-world 1.0.0	Published
> inventory auto product	inventory-auto-product 1.0.0	Published
> ItemService auto product	itemservice-auto-product 1.0.0	Published
> logistics auto product	logistics-auto-product 1.0.0	Published
> savings auto product	savings-auto-product 1.0.0	Published
> think-product	think-product 1.0.0	Published

[Publishing and managing products and APIs](#)

© Copyright IBM Corporation 2020

Figure 13-19. Manage published products in API Manager

You can manage the lifecycle states of a published product from the Manage option in API Manager. Click the catalog to open it. The list of published products is displayed.

Changes to the product lifecycle state can be made from the options ellipsis for a particular product.

The options from the Manage menu for published products in API Manager includes the lifecycle options to deprecate or retire the product and to edit the visibility and subscribers.

Other selections include replacing or superseding an existing product.

Remove a product from the catalog (1 of 2)

- From the Manage page, select the option to **Retire** the product

TITLE	NAME	STATE	
> financing auto product	financing-auto-product 1.0.0	Published	⋮
> hello-world	hello-world 1.0.0	Published	⋮
> inventory auto product	inventory-auto-product 1.0.0	Published	Deprecate
> ItemService auto product	itemservice-auto-product 1.0.0	Published	Retire
> logistics auto product	logistics-auto-product 1.0.0	Published	Replace

- Click the option to confirm



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-20. Remove a product from the catalog (1 of 2)

You remove a product from a catalog in two steps.

The first step is to retire the product. When you select the retire option, all associated APIs are taken offline and any subscriptions become inactive.

Remove a product from the catalog (2 of 2)

- The product is now in the retired lifecycle state

TITLE	NAME	STATE	⋮
> financing auto product	financing-auto-product 1.0.0	Published	⋮
> hello-world	hello-world 1.0.0	Retired	⋮

- Click the **Remove** option for the product

TITLE	NAME	STATE	⋮
> financing auto product	financing-auto-product 1.0.0	Published	⋮
> hello-world	hello-world 1.0.0	Retired	⋮
> inventory auto product	inventory-auto-product 1.0.0	Published	Re-stage
> ItemService auto product	itemservice-auto-product 1.0.0	Published	Edit visibility
> logistics auto product	logistics-auto-product 1.0.0	Published	Remove 

- Confirm removal
- The product is deleted from the catalog and now exists as a draft

Figure 13-21. Remove a product from the catalog (2 of 2)

The product is now in the retired state on the catalog. Select the Remove option for the product. Confirm the removal in the confirmation dialog that follows. The product is deleted from the catalog and is not displayed in the list of product for the catalog. The product now exists as a “draft” product and can be viewed and edited from the Develop APIs and products in API Manager.



Permissions for managing products

- From the Settings tab of the API Manager home page, select **Roles**

ROLES	
Administrator	⋮
API Administrator	⋮
Community Manager	⋮
Developer	⋮
Member	⋮
Owner	⋮
Owns and administers the API provider organization	
Member	Settings
	• View • Manage
Settings	• View • Manage

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-22. Permissions for managing products

By default, the catalog owner and administrator have all permissions to manage products in the Sandbox catalog. Other roles have view permissions by default.

You can edit and customize the permissions for lifecycle changes from the roles option of the settings tab in API Manager.



Information

In this course, you use the user with an owner role to manage the products and APIs in API Manager.

Unit summary

- Explain the concept of a plan, a product, and a catalog
- Explain the staging and publishing API lifecycle stages
- Define an API product and a plan
- Describe the steps to publish a product
- Describe the Developer Toolkit interface for managing products and APIs
- Explain the lifecycle states for products and APIs

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-23. Unit summary

Review questions

1. Which of these statements are true?
 - A. A product can be published to selected communities of application developer organizations
 - B. Plans within the product can be used to tailor access and visibility further
 - C. APIs become accessible when a product is published and made visible on the Developer Portal
 - D. All of the above

2. Which of these statements is *not* a lifecycle state?
 - A. Stage
 - B. Publish
 - C. Catalog
 - D. Retire



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-24. Review questions

Write your answers here:

- 1.

- 2.

Review answers



1. Which of these statements are true?
 - A. A product can be published to selected communities of application developer organizations
 - B. Plans within the product can be used to tailor access and visibility further
 - C. APIs become accessible when a product is published and made visible on the Developer Portal
 - D. All of the above

The answer is D.

2. Which of these statements is *not* a lifecycle state?
 - A. Stage
 - B. Publish
 - C. Catalog
 - D. Retire

The answer is C.

Exercise: Define and publish an API product

Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-26. Exercise: Define and publish an API product

Exercise objectives

- Create a product and plan in API Manager
- Add the APIs to the product
- Verify that the invoke URL for the inventory application routes to the Docker image
- Publish the product to the Sandbox catalog



Publishing and managing products and APIs

© Copyright IBM Corporation 2020

Figure 13-27. Exercise objectives

Unit 14. Subscribing and testing APIs

Estimated time

01:00

Overview

This unit explores the application developer user experience. In the API Connect architecture, the application developer creates an application that calls published APIs. To use APIs, an application developer creates an account in the Developer Portal. This unit explains how the application developer subscribes to a plan and tests API operations.

How you will check your progress

- Review question
- Lab exercise

Unit objectives

- Explain the role of application developers in calling published APIs
- Describe the Developer Portal self-registration process for development catalogs
- Explain how to add an application in the Developer Portal
- Describe the role of client ID and client secret for application identification
- Describe how to subscribe to an API plan
- Describe the subscription approval process
- Explain the test client features in the Developer Portal

Role of application developers

- Application developers discover and use APIs by using the Developer Portal
- When API providers publish an API, they can specify one or more consumer organizations, thus restricting visibility of the API
- Only application developers in the specified organizations can see the API on the Developer Portal and obtain application keys to access it
- A consumer organization might represent an individual or a group of application developers
- Application developers register their applications and subscribe to plans on the Developer Portal

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-2. Role of application developers

Application developers use the Developer Portal to discover and use APIs. API developers create and publish APIs. API developers belong to provider organizations.

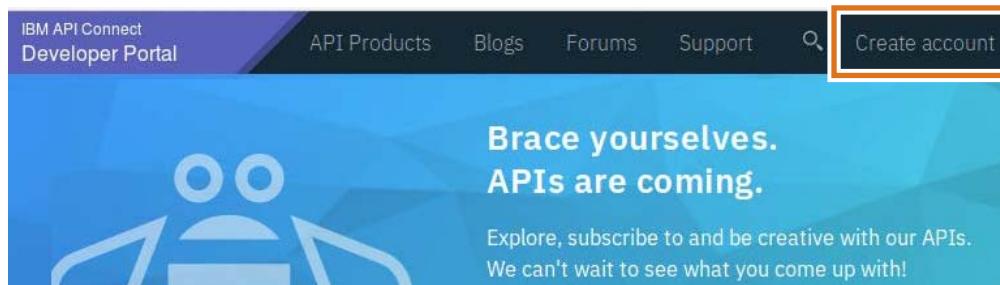
Application developers belong to consumer organizations.

When API providers publish APIs, they can restrict visibility of the APIs to one or more consumer organizations, or make the APIs visible to public or authenticated users.



Self-registration on the Developer Portal (1 of 5)

- Developers can do their own sign-up process when the *Self-service onboarding* is set to “enabled” for the Developer Portal in the API Manager interface
 - Generally used only for sandbox development catalogs
- Developer Portal home page includes a **Create account** link
- Application Developers can create their own user account and consumer organization on the Developer Portal



Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-3. Self-registration on the Developer Portal (1 of 5)

Developers can sign up to the Developer Portal without requiring an invitation from the API provider. Enable the self-service onboarding in the API Manager interface settings for the catalog. The Developer Portal then includes a link to create an account.

The self-sign up process is generally only enabled for sandbox development catalogs so that application developers can easily register, view, and subscribe to APIs.



Self-registration on the Developer Portal (2 of 5)

- The sign-up page is displayed after the Create Account is selected on the Developer Portal

1

API Developer Portal

Sign up

Sign up with Sandbox Catalog User Registry

Username *

Email address *

First Name *

Last Name *

Consumer organization *

2

Password *

.....

Password strength: Weak

Confirm password *

Your password meets the password policies required for this site

Q B Y A L

What code is in the image? *

Enter the characters shown in the image.

Sign up

Subscribing and testing APIs

© Copyright IBM Corporation 2020

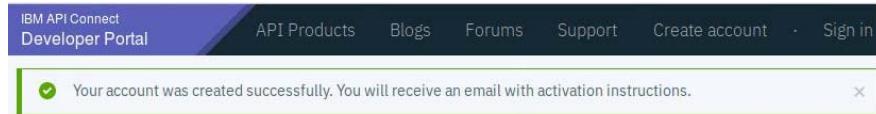
Figure 14-4. Self-registration on the Developer Portal (2 of 5)

The “Sign up” page is shown in two parts. The user creates an account with a user name, email address, given name, family name, and a consumer organization name that gets created when the account is created.

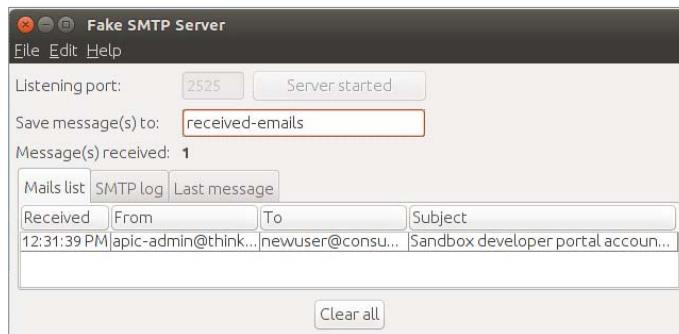


Self-registration on the Developer Portal (3 of 5)

- An activation link is sent to the email server



- User responds by opening the email and clicking the email link to activate the account in the Developer Portal



Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-5. Self-registration on the Developer Portal (3 of 5)

An email notification is displayed. An email message is sent to the email address that was specified during account creation. When the email is opened, a link is displayed in the body of the email message. Copying the link into a browser address activates the user account on the Developer Portal.



Self-registration on the Developer Portal (4 of 5)

- The account is activated on the Developer Portal



- User signs on to the Portal with the same credentials that were specified during account creation

 A screenshot of the "API Developer Portal" sign-in page. The page title is "Sign in". It features a "Sign in with Sandbox Catalog User Registry" link and a "Continue with" section. In the "Continue with" section, there is a blue button labeled "admin". Below the sign-in form, there is a "Sign in" button. The "Username" field contains "newuser" and the "Password" field contains a redacted password.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

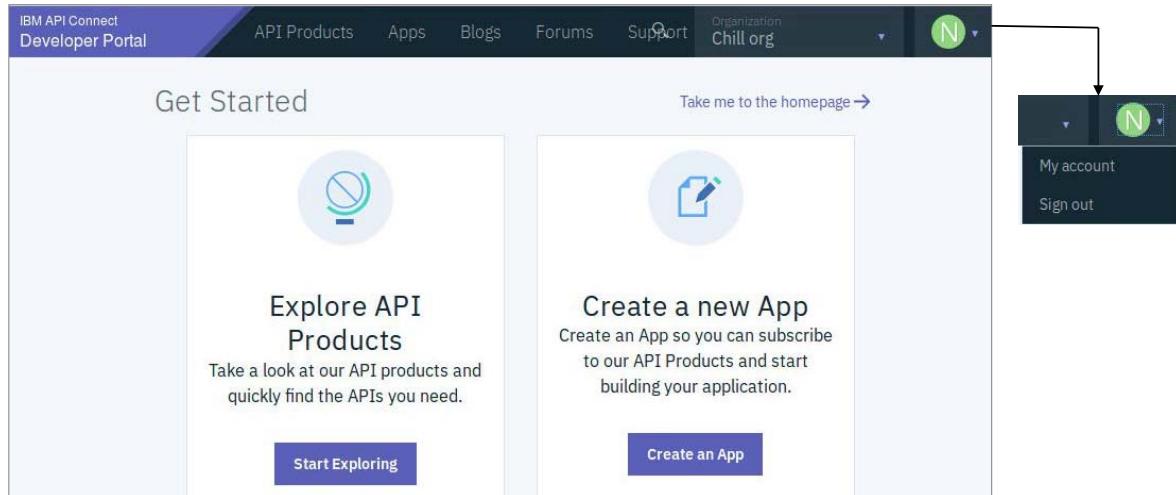
Figure 14-6. Self-registration on the Developer Portal (4 of 5)

The account is activated on the Developer Portal. The user can sign on to the Developer Portal with the same credentials that were specified during account creation.




Self-registration on the Developer Portal (5 of 5)

- User is signed on and can now manage the account from the My account option



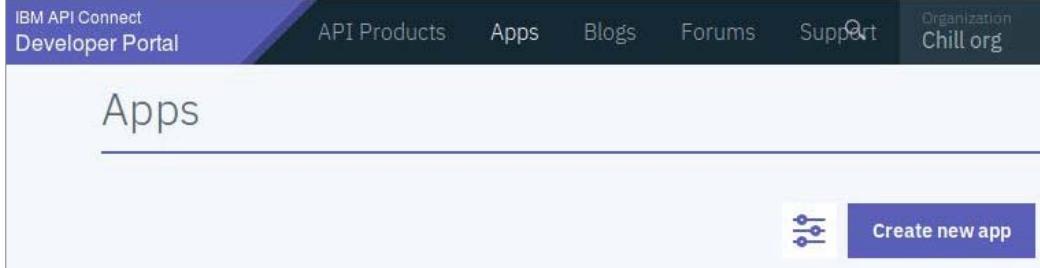
Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-7. Self-registration on the Developer Portal (5 of 5)

The user is authenticated and signed on to the Developer Portal.

The user can manage the account and organization from the list that is displayed below the user's icon.



The screenshot shows the IBM API Connect Developer Portal interface. At the top, there's a blue header bar with the text "IBM Training" on the left and the "IBM" logo on the right. Below the header is a main navigation bar with several tabs: "API Products", "Apps", "Blogs", "Forums", "Support", and "Organization". The "Organization" tab has a dropdown menu open, showing "Chill org" as the selected option. The main content area is titled "Apps". In the bottom right corner of this area, there's a prominent blue button with the text "Create new app" in white.

- Select **Apps** from the menu
- Then, click **Create new App**

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-8. Create an application (1 of 3)

The **Create new App** link is available on the **Apps** tab on the Developer Portal.



Create an application (2 of 3)

- Type the name in the Title field and an optional description
- Click **Submit**

A screenshot of a web-based application creation form titled "Create a new application".

- Title ***: A text input field containing "Think Application".
- Description**: A text input field containing "Latest Think App".
- Application OAuth Redirect URL(s)**: A section with a list box containing a single item "http://thinkapp.com". Below this is a button labeled "Add another item".
- Buttons**: Two buttons at the bottom: "Cancel" and "Submit".

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-9. Create an application (2 of 3)

Specify a title for the application.

Optionally specify a description and an OAuth redirect URL.

Then, click Submit to create the application.



Create an application (3 of 3)

- Type application is created
- Click the **Show** options to display the API key and secret

Application created successfully.

API Key and Secret

The API Key and Secret have been generated for your application.

Key

Show

Secret

Show

The Secret will only be displayed here one time. Please copy your API Secret and keep it for your records.

Continue

- Client ID and Client Secret are credentials that an application uses to identify itself

[Subscribing and testing APIs](#)

© Copyright IBM Corporation 2020

Figure 14-10. Create an application (3 of 3)

When an API operation is called, you can require that an application must provide either a client ID, or a client ID and client secret.

The identification requirements for calling an API are specified in the API security definitions in API Manager.

These requirements include supplying a client ID, client ID and client secret, or none.

Click the options to show the API key and secret. Save these values for APIs that require an API key and secret.



Subscribe an application to a Product plan (1 of 6)

- You can browse the available APIs at the end of the registration form and subscribe to a plan. Click the **available APIs** link

The screenshot shows the 'Think Application' interface. At the top, there are tabs for 'Dashboard' and 'Subscriptions'. The 'Subscriptions' tab is active. Below it, under 'Credentials', there are fields for 'Client ID' (redacted) and 'Client Secret' (redacted), with a 'Verify' button. Under 'Subscriptions', there is a table with columns 'PRODUCT' and 'PLAN'. A message 'No subscriptions found' is displayed, followed by a link 'i. Why not browse the available APIs?'. This link is highlighted with an orange rectangle.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-11. Subscribe an application to a Product plan (1 of 6)

The **available APIs** link is at the end of the application subscriptions page.

Clicking the **available APIs** link takes you to the list of Products and their associated APIs.



Subscribe an application to a Product plan (2 of 6)

- The list of Products is displayed
- Click the link for the Product

A screenshot of a web interface showing a list of products. At the top, there is a green circular icon with a white 'P' and the text "think-product 1.0.0". Below this, there is a horizontal line. Underneath the line, the text "inventory 1.0.0", "financing 1.0.0", and "logistics 1.0.0" is listed. At the bottom of the list, there is a navigation bar with buttons for "« First", "«", "1", and "2". The button "2" is highlighted with a blue box.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-12. Subscribe an application to a Product plan (2 of 6)

Click the link for the product to open the product and view the list of APIs and plans.



Subscribe an application to a Product plan (3 of 6)

- The Product APIs and plans are displayed
- Select the plan that you want to use
- The default plan is a rate-limited plan, while the gold plan is an unlimited plan

A screenshot of the IBM API Management interface. At the top, it shows the product name "think-product" version 1.0.0 with a 5-star rating. Below this, under the "APIs" section, there are three items: "inventory 1.0.0" (with a gear icon), "financing 1.0.0" (with a person icon), and "logistics 1.0.0" (with a truck icon). Under the "Plans" section, there are two options: "Default Plan" and "Gold Plan". The "Gold Plan" is highlighted with an orange border. Both plans have a "Subscribe" button below them. A small "View details" link is visible between the plans.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-13. Subscribe an application to a Product plan (3 of 6)

Here you see the APIs for the product that is named “think-product”.

In the example, you can select the default plan or gold plan. The gold plan is selected.

The gold plan displays a key icon that indicates that the plan requires approval.



Subscribe an application to a Product plan (4 of 6)

- Select the application that is to be subscribed to the plan
- Click **Select App**

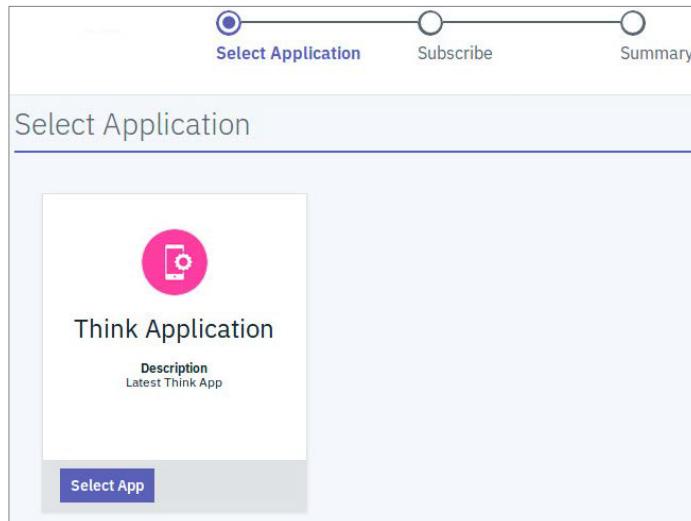


Figure 14-14. Subscribe an application to a Product plan (4 of 6)

Select the application that is being subscribed to the plan.



Subscribe an application to a Product plan (5 of 6)

- Confirm the subscription properties

Confirm Subscription

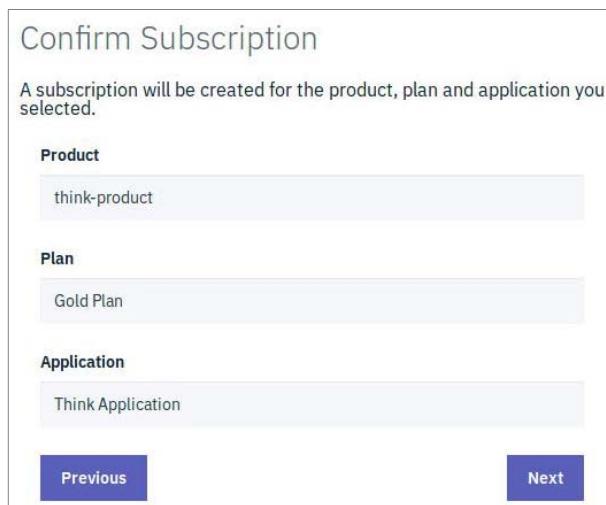
A subscription will be created for the product, plan and application you selected.

Product
think-product

Plan
Gold Plan

Application
Think Application

Previous **Next**



Subscribing and testing APIs

© Copyright IBM Corporation 2020

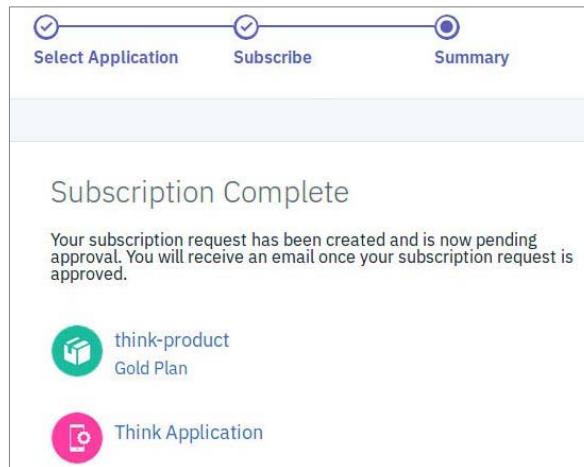
Figure 14-15. Subscribe an application to a Product plan (5 of 6)

Review the subscription values. Then, click Next to confirm and create the subscription.



Subscribe an application to a Product plan (6 of 6)

- The subscription is complete and the application is subscribed to the plan
- Since the gold plan was selected, the subscription request is pending approval



Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-16. Subscribe an application to a Product plan (6 of 6)

The subscription request is created and is now pending approval.

Since the gold plan is selected, the API provider must approve the subscription request in API Manager.

The screenshot shows the IBM API Connect Developer Portal interface. At the top, there's a blue header bar with the text "IBM Training" on the left and the "IBM" logo on the right. Below the header, the main content area has a title "Approval requests (1 of 3)". The navigation bar at the top includes links for "API Products", "Apps", "Blogs", "Forums", "Support", and "Organization Chill.org". The main content area is titled "Think Application". It has two tabs: "Dashboard" and "Subscriptions", with "Subscriptions" being the active tab. Under "Subscriptions", there are two sections: "Credentials" and "Subscriptions". The "Credentials" section contains fields for "Client ID" (with a redacted value) and "Client Secret" (with a redacted value), along with a "Verify" button. The "Subscriptions" section lists one item: "think-product (1.0.0) (Pending approval)" under the "PLAN" column, which is highlighted with an orange border. The "PRODUCT" column shows "Gold Plan". There are also three-dot and plus icons in the "Subscriptions" section.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-17. Approval requests (1 of 3)

If you go to the application in the Developer Portal, you see that the subscription is pending approval.



Approval requests (2 of 3)

- Approval list in API Manager
- From the Manage page for the catalog, click **Tasks** option
- The subscription request can be approved or declined

The screenshot shows the 'Tasks' page under 'Manage / Sandbox'. It has two tabs: 'Approval Tasks' (selected) and 'Requested Approvals'. A single task is listed:

- Subscription approval task for subscription of think-product:1.0.0 (Gold Plan) requested by newuser (chill-org)
- Subscription
- 12 minutes ago
- Buttons: Decline (light blue) and Approve (dark blue)

- When the task is completed, the list of approvals is updated

The screenshot shows the 'Tasks' page under 'Manage / Sandbox'. The 'Approval Tasks' tab is selected. A message box displays:

- ⚠ There are no tasks to be displayed

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-18. Approval requests (2 of 3)

The API producer owner can approve the subscription request in API Manager from the Tasks page of the catalog.



Approval requests (3 of 3)

- Verify that the application is subscribed to the plan in the Developer Portal

A screenshot of the IBM Cloud developer portal. The top navigation bar has "IBM Training" on the left and the IBM logo on the right. The main content area has a title "Think Application". Below it, there are two tabs: "Dashboard" and "Subscriptions", with "Subscriptions" being the active tab. Under "Subscriptions", there is a section titled "Credentials" which contains fields for "Client ID" (redacted) and "Client Secret" (redacted), with a "Verify" button. Below this is a section titled "Subscriptions" with a table:

PRODUCT	PLAN	⋮
think-product (1.0.0)	Gold Plan	⋮

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-19. Approval requests (3 of 3)

The page shows that the application is subscribed to the gold plan for the think-product.

The application can now call and test APIs.

Testing an API in the Developer Portal

- The API definition must include the statements:

```
x-ibm-configuration:  
  testable: true  
  enforced: true  
  cors:  
    enabled: true
```

The screenshot shows the IBM API Connect API Manager interface. On the left, there's a sidebar with icons for Home, API, Catalog, and Settings. The main area has a title bar 'IBM API Connect API Manager' and a sub-header 'Develop logistics 1.0.0'. Below this, there are three tabs: 'Design' (selected), 'Source', and 'Assemble'. The 'Lifecycle' tab is currently active. It displays the status 'Realized' under 'Lifecycle (optional)'. In the 'Definitions' section, several checkboxes are checked: 'Enforced', 'Testable', 'CORS', and 'Buffering'. The checkbox for 'Application authentication' is not checked.

[Subscribing and testing APIs](#)

© Copyright IBM Corporation 2020

Figure 14-20. Testing an API in the Developer Portal

For an API to be tested on the Developer Portal, the definition file for the API to be tested must include the statements:

```
x-ibm-configuration:  
  testable: true  
  enforced: true  
  cors:  
    enabled: true
```

Notice that the security requirements are also specified in the YAML source file.

In the example, a call to the API requires the application to authenticate with a client ID.



Testing an API in the Developer Portal (1 of 6)

- Open a Product from the **API Products** link
- Select the API to be tested

A screenshot of the IBM API Connect Developer Portal. At the top, there's a navigation bar with links for "API Products", "Apps", "Blogs", "Forums", "Support", and "Organization Chill.org". The main content area shows a product card for "think-product 1.0.0". Below the card, a list of APIs is shown: "inventory 1.0.0", "financing 1.0.0", and "logistics 1.0.0". The "logistics" item is highlighted with an orange rectangular box around it.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-21. Testing an API in the Developer Portal (1 of 6)

If the testable option is enabled for an API, then the API can be tested on the Developer Portal. After the product is selected, select the API that you want to test from the list of APIs.



Testing an API in the Developer Portal (2 of 6)

- Select the operation to be tested

The screenshot shows the IBM Developer Portal interface. At the top, it displays "All Products / think-product" and the "logistics" API version 1.0.0, which has a 5-star rating. Below this, there's a "Filter" button and a sidebar with "Overview" selected. The main content area is titled "Overview" and contains the following details:

- Download:** Open API Document
- Type:** REST
- Endpoint:** Production, Development: <https://apigw.think.ibm/think/sandbox/logistics>
- Security:**
 - clientIDHeader:** X-IBM-Client-Id apiKey located in header

A red box highlights the "GET /stores" operation in the sidebar.

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-22. Testing an API in the Developer Portal (2 of 6)

Select the API operation that you want to test in the Developer Portal.



Testing an API in the Developer Portal (3 of 6)

- Click the **Try it** tab on the page

A screenshot of the IBM API Management developer portal. At the top, there's a navigation bar with "IBM Training" and the IBM logo. Below that, the main content area shows an API named "logistics 1.0.0" with a 5-star rating. On the left, there's a sidebar with "Overview", "GET /shipping", and "GET /stores" (which is selected and highlighted in blue). In the center, there's a "Locate store near zip code" section with a "Try it" button that has an orange border. Below that, it says "GET" and "Production, Development: https://apigw.think.ibm/think/sandbox/logistics/stores". Under "Security", it shows "clientIdHeader" and "X-IBM-Client-Id" with the note "apiKey located in header".

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-23. Testing an API in the Developer Portal (3 of 6)

The details for the GET /stores operation is displayed on the page.



Testing an API in the Developer Portal (4 of 6)

- The Developer Portal automatically inserts the client ID from the application you created earlier

A screenshot of the IBM Developer Portal interface. On the left, there's a sidebar with a "Filter" button, "Overview" section (with "GET /shipping" and "GET /stores" listed, where "GET /stores" is selected), and a "Definitions" section. The main panel shows an API endpoint titled "Locate store near zip code". Under "Details", it shows a "Try it" button, a "GET" method, "Production, Development" environment, and the URL "https://apigw.think.ibm/think/sandbox/logistics/stores". Under "Security", there are sections for "Identification" and "Client ID". The "Client ID" dropdown is set to "Think Application".

Figure 14-24. Testing an API in the Developer Portal (4 of 6)

The Developer Portal automatically inserts the client ID that gets generated when the application was created.



Testing an API in the Developer Portal (5 of 6)

- Type the required parameters. Then, click **Send**

The screenshot shows the IBM API Testing interface. On the left, there's a sidebar with a "Filter" button, "Overview" (selected), "GET /shipping", and "GET /stores" (selected). Below that is a "Definitions" section. The main area is titled "Parameters". Under "Header", "Accept" is set to "application/json". Under "Content-Type", it is also set to "application/json". Under "Query", there is a field labeled "zip*" with the value "94126". At the bottom right are two buttons: "Reset" and "Send".

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-25. Testing an API in the Developer Portal (5 of 6)

Type the required parameters for the operation. Click **Send** to call the API operation on the gateway.

Testing an API in the Developer Portal (6 of 6)

- Review the request and response

Request	<pre>GET https://apigw.think.ibm/think/sandbox/1 ogistics/stores?zip=94126 Headers: Accept: application/json X-IBM-Client-Id: e590944d403b8097f536474699 ad04d5</pre>
Response	<pre>Code: 200 OK Headers: content-type: text/xml x-global-transaction-id: 196c55655c9e87cd00 0006e3 x-ratelimit-limit: name=default,100; x-ratelimit-remaining: name=default,99; {"maps_link":"https://www.openstreetmap.org /#map=16/55.6848895758499/21.1675253702887" }</pre>

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-26. Testing an API in the Developer Portal (6 of 6)

Examples of the request and response messages are shown on the page.



Status of applications in API Manager

- List of applications displayed in API Manager

A screenshot of a web-based application interface titled "Manage / Sandbox Applications". At the top right is a blue "Add ▾" button. Below it is a table with four columns: "TITLE", "APPLICATION TYPE", "CONSUMER ORGANIZATION", and "STATE". There are three rows of data:

TITLE	APPLICATION TYPE	CONSUMER ORGANIZATION	STATE
> think-application	Production	chill-org	Enabled
> think-application	Production	publish	Enabled
> sandbox-test-app	Production	sandbox-test-org	Enabled

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-27. Status of applications in API Manager

The API provider can query the list of enabled application from the Manage option for the catalog in API Manager.

Unit summary

- Explain the role of application developers in calling published APIs
- Describe the Developer Portal self-registration process for development catalogs
- Explain how to add an application in the Developer Portal
- Describe the role of client ID and client secret for application identification
- Describe how to subscribe to an API plan
- Describe the subscription approval process
- Explain the test client features in the Developer Portal

Review questions

1. True or False: Self-registration can be disabled for the Developer Portal of a development (sandbox) catalog
2. Which one of the following options is required to enforce subscription approvals on the Developer Portal?
 - A. Lifecycle approvals must be enabled
 - B. Self-service onboarding must be enabled
 - C. Approvals for product lifecycle state changes must be enabled
 - D. Approvals must be enabled for the plan to which the application subscribes



Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-29. Review questions

Write your answers here:

- 1.
- 2.

Review answers

1. True or False: Self-registration can be disabled for the Developer Portal of a development (sandbox) catalog.
The answer is True.

2. Which one of the following options is required to enforce subscription approvals on the Developer Portal?
 - A. Lifecycle approvals must be enabled
 - B. Self-service onboarding must be enabled
 - C. Approvals for product lifecycle state changes must be enabled
 - D. Approvals must be enabled for the plan to which the application subscribes.

The answer is D.



Exercise: Subscribe and test APIs

Subscribing and testing APIs

© Copyright IBM Corporation 2020

Figure 14-31. Exercise: Subscribe and test APIs

Exercise objectives



- Review the consumer organizations of the Sandbox catalog in API Manager
- Review the portal settings for the Sandbox catalog
- Sign on to the Sandbox catalog Developer Portal as the owner of the consumer organization
- Register an application in the Developer Portal
- Review the client ID and client secret values
- Test API operations in the Developer Portal
- Test all the APIs that you created with a web-based consumer application

Figure 14-32. Exercise objectives

Unit 15. Course summary

Estimated time

00:15

Overview

This unit summarizes the course and provides information for future study.

Unit objectives

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

[Course summary](#)

© Copyright IBM Corporation 2020

Figure 15-1. Unit objectives

Course objectives

- Create APIs in API Connect
- Implement APIs with the LoopBack Node.js framework
- Create message processing policies that transform API requests and responses
- Authorize client API requests with security definitions
- Enforce an OAuth flow with an OAuth 2.0 API security
- Publish, and test APIs on the API Connect cloud

[Course summary](#)

© Copyright IBM Corporation 2020

Figure 15-2. Course objectives

To learn more on the subject

- IBM Training website:
www.ibm.com/training
- IBM API Connect learning journeys:
https://www-03.ibm.com/services/learning/ites.wss/zz-en?pageType=journey_description&journeyId=itns_185&tag=o-itns-01-02
- API Connect: IBM Developer Center
<https://developer.ibm.com/apiconnect/>
- IBM Knowledge Center for API Connect
https://www.ibm.com/support/knowledgecenter/SSMNED_2018/mapfiles/getting_started.html

Course summary

© Copyright IBM Corporation 2020

Figure 15-3. To learn more on the subject

Enhance your learning with IBM resources

Keep your IBM Cloud skills up-to-date

- IBM offers resources for:
 - Product information
 - Training and certification
 - Documentation
 - Support
 - Technical information



- To learn more, see the IBM Cloud Education Resource Guide:
 - www.ibm.biz/CloudEduResources

Course summary

© Copyright IBM Corporation 2020

Figure 15-4. Enhance your learning with IBM resources

Earn an IBM Badge

- Completing this course prepares you to take an IBM Badge test
- Use IBM Badges to share verified proof of your IBM credentials
- Find your Badge test on this site:
 - <https://www.ibm.com/services/learning/ites.wss/zz-en?pageType=badgesearch>
 - Search: *IBM API Connect API Developer*

Course summary

© Copyright IBM Corporation 2020

Figure 15-5. Earn an IBM Badge

Unit summary

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

Course summary

© Copyright IBM Corporation 2020

Figure 15-6. Unit summary

Appendix A. List of abbreviations

A

API application programming interface

B

C

CLI command-line interface

CORS Cross-Origin Resource Sharing

CPU central processing unit

D

DB database

DMZ demilitarized zone

E

EMS event management services

ESB enterprise service bus

F

G

H

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS HTTP over SSL

I

IBM International Business Machines Corporation

ICAP Internet Content Adaptation Protocol

IMS IBM Information Management System

IP Internet Protocol

IT information technology

J

JSON JavaScript Object Notation

JWK JSON Web Token key

JWT JSON Web Token

K**L**

LDAP	Lightweight Directory Access Protocol
LTPA	Lightweight Third Party Authentication

M

MB	megabyte
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MQ	Message Queue

N

NPM	node package manager
------------	----------------------

O

OAI	Open API Initiative
OS	operating system

P**Q****R**

REST	Representational State Transfer
-------------	---------------------------------

S

SAML	Security Assertion Markup Language
SDK	software development kit
SHA	secure hash algorithm
SOA	service-oriented architecture
SOAP	A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. Usage note: SOAP is not an acronym; it is a word in itself (formerly an acronym for Simple Object Access Protocol)
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer

T

TLS	Transport Layer Security
------------	--------------------------

U

UI	user interface
UML	Unified Modeling Language
UNIX	Uniplexed Information and Computing System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format

V

VM	virtual machine
-----------	-----------------

W

WS	web service
WSDL	Web Services Description Language

X

XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Y

YAML	Yet Another Markup Language
-------------	-----------------------------

Z

