

## Course Guide

# Creating, Publishing, and Securing APIs with IBM API Connect

Course code WD508 / ZD508 ERC 1.0



## December 2017 edition

### Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

### Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

**© Copyright International Business Machines Corporation 2017.**

**This document may not be reproduced in whole or in part without the prior written permission of IBM.**

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Trademarks .....</b>	<b>xiii</b>
<b>Course description .....</b>	<b>xiv</b>
<b>Agenda .....</b>	<b>xvi</b>
<b>Unit 1. IBM API Connect V5 overview.....</b>	<b>1-1</b>
How to check online for course material updates .....	1-2
Unit objectives .....	1-3
1.1. API Connect overview .....	1-4
API Connect overview .....	1-5
Topics .....	1-6
What is API Connect? .....	1-7
API Connect: Simplified and comprehensive API foundation .....	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
Monetize APIs .....	1-14
Create, run, manage, and secure .....	1-15
API Connect: Grows with your business needs .....	1-16
Key differences between API Connect offerings .....	1-17
What is the API Connect Cloud? .....	1-19
API Connect: Installation and configuration .....	1-20
API Connect: Runtime architecture (1 of 2) .....	1-21
API Connect: Runtime architecture (2 of 2) .....	1-22
API Connect: Developer Portal .....	1-23
API Connect: Development architecture .....	1-24
API Connect: Components by feature .....	1-25
Components and roles .....	1-26
API runtime message flow .....	1-28
Scenario 1: Proxy existing APIs .....	1-29
Scenario 2: Implement APIs .....	1-30
1.2. Gateway options.....	1-31
Gateway options .....	1-32
Topics .....	1-33
API gateway options .....	1-34
Gateway features .....	1-35
Gateway capabilities (1 of 2) .....	1-36
Gateway capabilities (2 of 2) .....	1-38
What is wrong with this picture? .....	1-40
You need an API gateway .....	1-41
1.3. Software and hardware prerequisites.....	1-42
Software and hardware prerequisites .....	1-43
Topics .....	1-44
API Connect: Requirements .....	1-45
Unit summary .....	1-46
Review questions .....	1-47
Review answers .....	1-48

<b>Unit 2. API Connect Developer Toolkit.....</b>	<b>2-1</b>
Unit objectives .....	2-2
What is the IBM API Connect toolkit? .....	2-3
How do you install the toolkit? .....	2-4
Step 1: Install and configure the prerequisite software .....	2-5
Review the supported level for Node.js .....	2-7
Step 2: Install the Node runtime environment and NPM .....	2-8
Step 3: Install the apiconnect Node package .....	2-9
Step 4: Verify the toolkit installation .....	2-10
Installation troubleshooting: User permissions .....	2-11
Installation troubleshooting: Xcode .....	2-12
Installation troubleshooting: sqlite .....	2-13
Upgrade or reinstall the Developer Toolkit .....	2-14
API Connect Toolkit: The apic command .....	2-15
API Connect Toolkit: The API designer web application .....	2-16
How to create APIs in API Connect .....	2-17
Unit summary .....	2-18
Review questions .....	2-19
Review answers .....	2-20
Demonstration: Installing and verifying the Developer Toolkit .....	2-21
Demonstration objectives .....	2-22
<b>Unit 3. Creating an API definition.....</b>	<b>3-1</b>
Unit objectives .....	3-2
What is an API definition? .....	3-3
Open API: REST API interface standard .....	3-4
OpenAPI definition structure .....	3-5
How do you create an API definition? .....	3-7
OpenAPI definition: Name, version, and licensing .....	3-8
Parts of an REST API definition .....	3-9
Parts of an REST API operation .....	3-10
Review: HTTP methods in REST architecture .....	3-11
Example: Savings plan estimate .....	3-12
Example: GET /Plans/estimate .....	3-13
Step 1: Define a path, and operation .....	3-14
Step 2: Define the input parameters and response message .....	3-15
Step 3: Define the result data type .....	3-16
Example: POST/Plans/estimate .....	3-17
Step 1: Create a POST operation in the existing path .....	3-18
Step 2: Define the input parameter in the message body .....	3-19
Step 3: Define the plan schema data type .....	3-20
API definition: Source view .....	3-21
Example: savings_1.0.0.yaml .....	3-22
IBM extensions to the OpenAPI definition format .....	3-23
Extensions: Lifecycle settings .....	3-24
Extensions: Environment-specific properties .....	3-25
Extensions: Message processing policy assembly .....	3-26
Example: The invoke policy .....	3-27
API templates .....	3-28
Example: Default API definition template .....	3-29
Unit summary .....	3-30
Review questions .....	3-31
Review answers .....	3-32
Exercise: Creating and publishing an API in API Designer .....	3-33
Exercise objectives .....	3-34

<b>Unit 4. Defining APIs that call REST and SOAP services .....</b>	<b>4-1</b>
Unit objectives .....	4-2
What types of APIs can you define? .....	4-3
Scenarios .....	4-4
4.1. Proxy an existing REST API.....	4-5
Proxy an existing REST API .....	4-6
Topics .....	4-7
Overview: Proxy an existing REST API .....	4-8
Scenario one: Define a REST API .....	4-9
Scenario one: Gateway forwards REST API requests .....	4-10
Step 1: Create a REST API definition in API Designer .....	4-11
Step 2: Define the base path and target endpoint .....	4-12
Step 3: Define the API operations .....	4-13
Step 4: Configure the gateway to call the REST application .....	4-14
Step 5: Test the API definition .....	4-15
4.2. Proxy an existing SOAP API.....	4-16
Proxy an existing SOAP API .....	4-17
Topics .....	4-18
Overview: Proxy an existing SOAP API .....	4-19
Scenario two: Define a SOAP API .....	4-20
Scenario two: Gateway forwards SOAP API requests .....	4-21
Step 1: Review the existing SOAP service .....	4-22
Step 2: Create a SOAP API definition in API Designer .....	4-23
Step 3: Define the metadata and target endpoint .....	4-24
Step 4: Review the host, base path, and media type .....	4-25
Step 5: Review the API operations .....	4-26
Step 6: Review the request and response type definitions .....	4-27
Example: The item XML schema type .....	4-28
Example: The item API schema definition .....	4-29
Step 7: Review the proxy policy in the assembly flow .....	4-30
Step 8: Start the toolkit Docker DataPower gateway .....	4-31
Step 9: Test the SOAP API in the Designer toolkit .....	4-32
Step 10: Response message from the SOAP API call .....	4-33
Unit summary .....	4-34
Review questions .....	4-35
Review answers .....	4-36
Exercise: Defining an API that calls an existing SOAP service .....	4-37
Exercise objectives .....	4-38
<b>Unit 5. Implementing APIs with the LoopBack framework .....</b>	<b>5-1</b>
Unit objectives .....	5-2
5.1. Introduction to the LoopBack framework .....	5-3
Introduction to the LoopBack framework .....	5-4
Topics .....	5-5
API terminology .....	5-6
Example: Product inventory and price calculator .....	5-7
Message flows in IBM API Connect .....	5-8
What is LoopBack? .....	5-9
Implement API with a Node.js LoopBack application .....	5-10
5.2. Implement an API with a LoopBack application .....	5-11
Implement an API with a LoopBack application .....	5-12
Topics .....	5-13
Instructor demonstration .....	5-14
Step 1: Generate a LoopBack application .....	5-15
Step 2a: Define a data source .....	5-16
Step 2b: Configure the data source .....	5-17

Step 3: Create models, properties, and relationships .....	5-19
Step 4: Implement API logic .....	5-20
Step 5: Update the API definition file .....	5-22
Step 6: Start the API Designer application .....	5-23
Step 7: Review the API definition file with API Designer .....	5-24
Step 8a: Review the data source in the API Designer .....	5-25
Step 8b: Test the data source connector in the API Designer .....	5-26
Step 9: Start the LoopBack application and micro gateway .....	5-27
Step 10: Test the API with the API Explorer .....	5-28
Example: Test API operations in Explorer .....	5-29
Stop the gateway and application .....	5-30
Relationship between API gateway and LoopBack .....	5-31
Unit summary .....	5-32
Review questions .....	5-33
Review answers .....	5-34
Exercise: Creating a LoopBack application .....	5-35
Exercise objectives .....	5-36
<b>Unit 6. LoopBack models, properties, and relationships .....</b>	<b>6-1</b>
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: Remote methods and hooks .....	6-10
LoopBack framework: Non-database connectors .....	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
Example: Hello-world sample application .....	6-18
Example: Hello-world message model .....	6-19
How to implement an API with the LoopBack framework .....	6-20
Next steps after implementing the API .....	6-21
6.2. LoopBack models, properties, and relationships .....	6-22
LoopBack models, properties, and relationships .....	6-23
Topics .....	6-24
Steps: LoopBack application, models, properties, relations .....	6-25
Generate a LoopBack application with apic loopback .....	6-26
Example: Create a LoopBack application .....	6-27
LoopBack application: Directory structure .....	6-28
Where does LoopBack store model information? .....	6-29
Example: Create a model and properties .....	6-30
LoopBack model inheritance .....	6-31
LoopBack properties: Data types .....	6-32
Example: Model definition file .....	6-33
Example: Model script file .....	6-34
How do you define a relationship between models? .....	6-35
Example: Create model relations .....	6-36
Example: Relations in the model definition file .....	6-37
Model relation types .....	6-38
Unit summary .....	6-39

Review questions .....	6-40
Review answers .....	6-41
<b>Unit 7. Defining data sources with connectors .....</b>	<b>7-1</b>
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
Example: Create a memory data source .....	7-10
Example: Data sources configuration file .....	7-11
Example: Model configuration file .....	7-12
Database transactions .....	7-13
Reference: Transaction isolation levels .....	7-14
Non-database connectors .....	7-16
7.2. Generating LoopBack models from discovery and introspection .....	7-17
Generating LoopBack models from discovery and introspection .....	7-18
Topics .....	7-19
Model discovery and introspection .....	7-20
Generate models from relational databases .....	7-21
API Designer: Discover Models and Update Schema .....	7-22
API Designer: Select properties to generate .....	7-23
Model discovery: What files does it generate? .....	7-24
Example: Review the item MySQL database table .....	7-25
Example: Generated item.json model definition file .....	7-26
Create model instances from unstructured data .....	7-27
Example: Create a model from introspection (1 of 2) .....	7-28
Example: Create a model from introspection (2 of 2) .....	7-29
Unit summary .....	7-30
Review questions .....	7-31
Review answers .....	7-32
Exercise: Defining LoopBack data sources .....	7-33
Exercise objectives .....	7-34
<b>Unit 8. Implementing remote methods and event hooks .....</b>	<b>8-1</b>
Unit objectives .....	8-2
8.1. LoopBack remote methods and hooks .....	8-3
LoopBack remote methods and hooks .....	8-4
Topics .....	8-5
LoopBack framework: Remote methods and hooks .....	8-6
Steps: LoopBack remote methods and hooks .....	8-7
Adding logic to models .....	8-8
What is a remote method? .....	8-9
Example: Remote method .....	8-10
What is a remote hook? .....	8-11
Example: Remote hook .....	8-12
What is an operation hook? .....	8-13
Types of operation hooks .....	8-14
Which operations trigger operation hooks? .....	8-15
Example: Access operation hook .....	8-16
Example: After save operation hook .....	8-17
When do you implement remote methods? .....	8-18
When do you implement remote hooks? .....	8-19

When do you implement operation hooks?	8-20
Unit summary	8-21
Review questions	8-22
Review answers	8-23
Exercise: Implementing event-driven functions with remote and operation hooks	8-24
Exercise objectives	8-25
<b>Unit 9. Assembling message processing policies</b>	<b>9-1</b>
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	9-11
Example one: Forward an API call with the invoke policy	9-12
Example one: Forward an API call with the proxy policy	9-13
What is the difference between invoke and proxy policies?	9-14
Example two: Switch case by API operation	9-15
Example three: Map multiple API calls into a response	9-16
Example: Define input parameters in a map policy	9-17
Example: Map API results into a single response message	9-18
Example four: Transform REST to SOAP	9-19
Example: Define input parameters in a map policy	9-20
Example: Map REST parameters to a SOAP request	9-21
What is the message payload?	9-22
Example five: Validate properties in an HTTP message	9-23
Example six: Store message payload in API analytics	9-24
Example seven: JSON Web Tokens	9-25
Example: Encoded JSON Web Token (1 of 3)	9-26
Example: Encoded JSON Web Token (2 of 3)	9-27
Example: Encoded JSON Web Token (3 of 3)	9-28
Example: Generating a JSON Web Token (1 of 2)	9-29
Example: Generating a JSON Web Token (2 of 2)	9-30
User-defined policies	9-31
Example: writeToDatapowerLog policy	9-32
Example: writeToDatapowerLog custom policy	9-33
Assembly palette: Logic constructs	9-34
Assembly palette: API policies for invocation and code	9-35
Assembly palette: API policies for message mapping	9-36
Assembly palette: API policies for security	9-37
Unit summary	9-38
Review questions	9-39
Review answers	9-40
Exercise: Assembling message processing policies	9-41
Exercise objectives	9-42
<b>Unit 10. Declaring client authorization requirements</b>	<b>10-1</b>
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
Enforcing security definitions . . . . .	10-13
Rules for defining client ID and client secret . . . . .	10-14
Example: Client ID and client secret in the message header . . . . .	10-15
10.3. Authenticate clients with HTTP basic authentication . . . . .	10-16
Authenticate clients with HTTP basic authentication . . . . .	10-17
Topics . . . . .	10-18
Verifying identity with HTTP basic authentication . . . . .	10-19
Example: Storing credentials in HTTP request header . . . . .	10-20
Example: Basic authentication security definition . . . . .	10-21
Message confidentiality with Transport Layer Security . . . . .	10-22
10.4. Introduction to OAuth 2.0 . . . . .	10-23
Introduction to OAuth 2.0 . . . . .	10-24
Topics . . . . .	10-25
What is OAuth? . . . . .	10-26
Example: Allow third-party access to shared resources . . . . .	10-28
Example: Third-party access to inventory API resources . . . . .	10-29
Before OAuth: Sharing user passwords . . . . .	10-30
OAuth flow sequence . . . . .	10-31
OAuth Step 1: Resource owner requests access . . . . .	10-32
OAuth Step 2: OAuth client redirection to owner . . . . .	10-33
OAuth Step 3: Authenticate owner with authorization server . . . . .	10-34
OAuth Step 4: Ask resource owner to grant access to resources . . . . .	10-35
OAuth Step 5: Resource owner grants client access to resources . . . . .	10-36
OAuth Step 6: Authorization server sends authorization grant code to client . . . . .	10-37
OAuth Step 7: Client requests access token from authorization server . . . . .	10-38
OAuth Step 8: Authorization server sends authorization token to client . . . . .	10-39
OAuth Step 9: OAuth client sends access token to resource server . . . . .	10-40
OAuth Step 10: Resource server grants access to OAuth client . . . . .	10-41
Unit summary . . . . .	10-42
Review questions . . . . .	10-43
Review answers . . . . .	10-44
<b>Unit 11. Creating an OAuth 2.0 Provider . . . . .</b>	<b>11-1</b>
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 2.0 security definition? . . . . .	11-5
What is an OAuth 2.0 Provider? . . . . .	11-6
11.1. Create an OAuth 2.0 Provider . . . . .	11-7
Create an OAuth 2.0 Provider . . . . .	11-8
Topics . . . . .	11-9
Step 1: Create an OAuth 2.0 Provider . . . . .	11-10
Step 2: Configure the OAuth 2.0 Provider settings . . . . .	11-11
OAuth 2.0 Provider API: Client types . . . . .	11-12
OAuth 2.0 Provider API: OAuth flow and grant types . . . . .	11-13
Step 3: Authentication and authorization settings . . . . .	11-14
Step 4: Token settings . . . . .	11-15
OAuth 2.0 Provider API: Token settings . . . . .	11-16
OAuth 2.0 Provider: Metadata . . . . .	11-17
Example: Metadata that is stored in OAuth response message . . . . .	11-18

Example: Metadata that is stored in OAuth access token .....	11-19
How to: Add a token introspection endpoint .....	11-20
How to: Add a metadata URL to the OAuth 2.0 Provider .....	11-21
Example: Specify an external OAuth metadata URL .....	11-22
<b>11.2. Secure an API with OAuth 2.0 authorization .....</b>	<b>11-23</b>
Secure an API with OAuth 2.0 authorization .....	11-24
Topics .....	11-25
Step 1: Create an OAuth 2.0 security definition .....	11-26
Step 2: Create an OAuth 2.0 security definition .....	11-27
Unit summary .....	11-28
Review questions .....	11-29
Review answers .....	11-30
Exercise: Declaring an OAuth 2.0 Provider and security requirement .....	11-31
Exercise objectives .....	11-32
<b>Unit 12. Deploying an API to a Docker container .....</b>	<b>12-1</b>
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
Terminology: Service and swarm .....	12-26
Define the service .....	12-27
Example of initializing a Docker swarm .....	12-28
Deploy the service to a Docker swarm .....	12-29
Unit summary .....	12-30
Review questions .....	12-31
Review answers .....	12-32
Exercise: Deploying an API implementation to a container runtime .....	12-33
Exercise objectives .....	12-34
<b>Unit 13. Staging, publishing, and deploying an API product .....</b>	<b>13-1</b>
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 .....	13-6
Add an API .....	13-7

Add existing APIs to a product . . . . .	13-8
Catalogs . . . . .	13-9
Publish a plan and API . . . . .	13-10
Publish a plan and API result . . . . .	13-11
Publish from the terminal: Preparation (1 of 2) . . . . .	13-12
Publish from the terminal: Preparation (2 of 2) . . . . .	13-13
Publish a LoopBack project from the terminal . . . . .	13-14
Manage published products in API Manager . . . . .	13-15
Options on the Manage menu for a published product . . . . .	13-16
Lifecycle of products and API resources . . . . .	13-17
Republish a product version . . . . .	13-18
Product YAML file: Visibility section . . . . .	13-19
Stage and publish a previously published product (1 of 2) . . . . .	13-20
Stage and publish a previously published product (2 of 2) . . . . .	13-21
Create a version of a product in the API Designer . . . . .	13-22
New product version result in the API Designer . . . . .	13-23
Permissions for managing catalogs . . . . .	13-24
Product categories (1 of 2) . . . . .	13-25
Product categories (2 of 2) . . . . .	13-26
Publish the product . . . . .	13-27
Enable the Developer Portal to display categories (1 of 2) . . . . .	13-28
Enable the Developer Portal to display categories (2 of 2) . . . . .	13-29
Categories are displayed in the Developer Portal . . . . .	13-30
Category tags in the Developer Portal . . . . .	13-31
Unit summary . . . . .	13-32
Review questions . . . . .	13-33
Review answers . . . . .	13-34
Exercise: Defining and publishing an API product . . . . .	13-35
Exercise objectives . . . . .	13-36
<b>Unit 14. Subscribing and testing APIs . . . . .</b>	<b>14-1</b>
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
Register an application (1 of 3) . . . . .	14-9
Register an application (2 of 3) . . . . .	14-10
Register 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 7) . . . . .	14-22
Testing an API in the Developer Portal (2 of 7) . . . . .	14-23
Testing an API in the Developer Portal (3 of 7) . . . . .	14-24
Testing an API in the Developer Portal (4 of 7) . . . . .	14-25
Testing an API in the Developer Portal (5 of 7) . . . . .	14-26

Testing an API in the Developer Portal (6 of 7) . . . . .	14-27
Testing an API in the Developer Portal (7 of 7) . . . . .	14-28
Product catalog settings in API Manager (1 of 2) . . . . .	14-29
Product catalog settings in API Manager (2 of 2) . . . . .	14-30
Registration approvals (1 of 2) . . . . .	14-31
Registration approvals (2 of 2) . . . . .	14-32
Unit summary . . . . .	14-33
Review questions . . . . .	14-34
Review answers . . . . .	14-35
Exercise: Subscribing and testing APIs . . . . .	14-36
Exercise objectives . . . . .	14-37
<b>Unit 15. Troubleshooting . . . . .</b>	<b>15-1</b>
Unit objectives . . . . .	15-2
Internal communications at configuration and run time . . . . .	15-3
Running the Consumer web application (1 of 4) . . . . .	15-5
Running the Consumer web application (2 of 4) . . . . .	15-6
Running the Consumer web application (3 of 4) . . . . .	15-7
Running the Consumer web application (4 of 4) . . . . .	15-8
Calls made by the Consumer application . . . . .	15-9
Verify that Call #1 and #2 worked . . . . .	15-10
Verify that an OAuth token is sent in Call #3 . . . . .	15-11
Example: Troubleshooting the inventory application . . . . .	15-12
DataPower sign-on . . . . .	15-13
DataPower logs . . . . .	15-14
Call the API directly on localhost . . . . .	15-15
Call the API on the Docker container . . . . .	15-16
Examine the API properties . . . . .	15-17
Examine the API target runtime . . . . .	15-18
Verify that the product and APIs are published . . . . .	15-19
Test the API on the Developer Portal . . . . .	15-20
Example: Inventory call failed (1 of 4) . . . . .	15-21
Example: Inventory call failed (2 of 4) . . . . .	15-22
Example: Inventory call failed (3 of 4) . . . . .	15-23
Example: Inventory call failed (4 of 4) . . . . .	15-24
Working Consumer app . . . . .	15-25
Unit summary . . . . .	15-26
Review questions . . . . .	15-27
Review answers . . . . .	15-28
Exercise: Troubleshooting the case study . . . . .	15-29
Exercise objectives . . . . .	15-30
<b>Unit 16. Course summary . . . . .</b>	<b>16-1</b>
Unit objectives . . . . .	16-2
Course objectives . . . . .	16-3
To learn more on the subject . . . . .	16-4
Enhance your learning with IBM resources . . . . .	16-5
Unit summary . . . . .	16-6
Course completion . . . . .	16-7
<b>Appendix A. List of abbreviations . . . . .</b>	<b>A-1</b>

---

# 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

## Creating, Publishing, and Securing APIs with IBM API Connect

**Duration: 4 days**

### Purpose

This course teaches you how to create, publish, and secure APIs with IBM API Connect V5.0.8. You learn how to use the Developer Toolkit to define an API interface according to the OpenAPI specification. You assemble message processing policies in the API Designer web application and define client authorization schemes, such as OAuth 2.0, in the API definition. You build a Node.js API application with the LoopBack framework. You also package, stage, and publish an API with a product and plan.

### 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 with the API Connect toolkit
- Implement APIs with the LoopBack Node.js framework
- Validate, filter, and transform API requests and responses with message processing policies
- Authorize client API requests with security definitions
- Enforce an OAuth flow with an OAuth 2.0 Provider API
- Stage, publish, and test APIs on the API Connect Cloud

### Contents

- IBM API Connect V5 overview
- API Connect developer toolkit
- Creating an API definition
- Exercise: Creating and publishing an API in API Designer
- Defining APIs that call REST and SOAP services

- Exercise: Defining an API that calls an existing SOAP service
- Implementing APIs with the LoopBack framework
- Exercise: Creating a LoopBack application
- LoopBack models, properties, and relationships
- Defining data sources with connectors
- Exercise: Defining LoopBack data sources
- Implementing remote methods and event hooks
- Exercise: Implementing event-driven functions with remote and operation hooks
- Assembling message processing policies
- Exercise: Assembling message processing policies
- Declaring client authorization requirements
- Creating an OAuth 2.0 Provider
- Exercise: Declaring an OAuth 2.0 Provider and security requirement
- Deploying an API to a Docker container
- Exercise: Deploying an API implementation to a container runtime
- Staging, publishing, and deploying an API product
- Exercise: Defining and publishing an API product
- Subscribing and testing APIs
- Exercise: Subscribing and testing APIs
- Troubleshooting
- Exercise: Troubleshooting the case study
- 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
- (01:15) Unit 1. IBM API Connect V5 overview**
- (01:00) Unit 2. API Connect Developer Toolkit**
- (01:15) Unit 3. Creating an API definition**
- (01:00) Exercise 1. Creating and publishing an API in API Designer
- (01:00) Unit 4. Defining APIs that call REST and SOAP services**
- (01:00) Exercise 2. Defining an API that calls an existing SOAP service

## Day 2

- (01:00) Unit 5. Implementing APIs with the LoopBack framework**
- (01:00) Exercise 3. Creating a LoopBack application
- (01:15) Unit 6. LoopBack models, properties, and relationships**
- (01:00) Unit 7. Defining data sources with connectors**
- (01:00) Exercise 4. Defining LoopBack data sources
- (00:45) Unit 8. Implementing remote methods and event hooks**
- (00:45) Exercise 5. Implementing event-driven functions with remote and operation hooks

## Day 3

- (01:30) Unit 9. Assembling message processing policies**
- (02:00) Exercise 6. Assembling message processing policies
- (01:00) Unit 10. Declaring client authorization requirements**
- (01:00) Unit 11. Creating an OAuth 2.0 Provider**
- (01:00) Exercise 7. Declaring an OAuth 2.0 Provider and security requirement

## Day 4

### **(00:45) Unit 12. Deploying an API to a Docker container**

(00:45) Exercise 8. Deploying an API implementation to a container runtime

### **(01:00) Unit 13. Staging, publishing, and deploying an API product**

(00:45) Exercise 9. Defining and publishing an API product

### **(01:00) Unit 14. Subscribing and testing APIs**

(01:00) Exercise 10. Subscribing and testing APIs

### **(00:30) Unit 15. Troubleshooting**

(01:15) Exercise 11. Troubleshooting the case study

### **(00:15) Unit 16. Course summary**

---

# Unit 1. IBM API Connect V5 overview

## Estimated time

01:15

## Overview

This unit explains the scope and purpose of IBM API Connect V5 from the perspective of an API developer. You review the role of APIs in the Cloud network and enterprise architecture. You also examine the API create, run, manage, and secure features in an API Connect solution.

## How you will check your progress

- Review questions

## How to check online for course material updates



**Note:** If your classroom does not have internet access, ask your instructor for more information.

### Instructions

1. Enter this URL in your browser:  
[ibm.biz/CloudEduCourses](http://ibm.biz/CloudEduCourses)
2. Find the product category for your course, and click the link to view all products and courses.
3. Find your course in the course list and then click the link.
4. The wiki page displays information for the course. If a course corrections document is available, this page is where it is found.
5. If you want to download an attachment, such as a course corrections document, click the **Attachments** tab at the bottom of the page.  
  
Comments (0)   Versions (1)   **Attachments (1)**   About
6. To save the file to your computer, click the document link and follow the prompts.

Figure 1-1. How to check online for course material updates

## Unit objectives

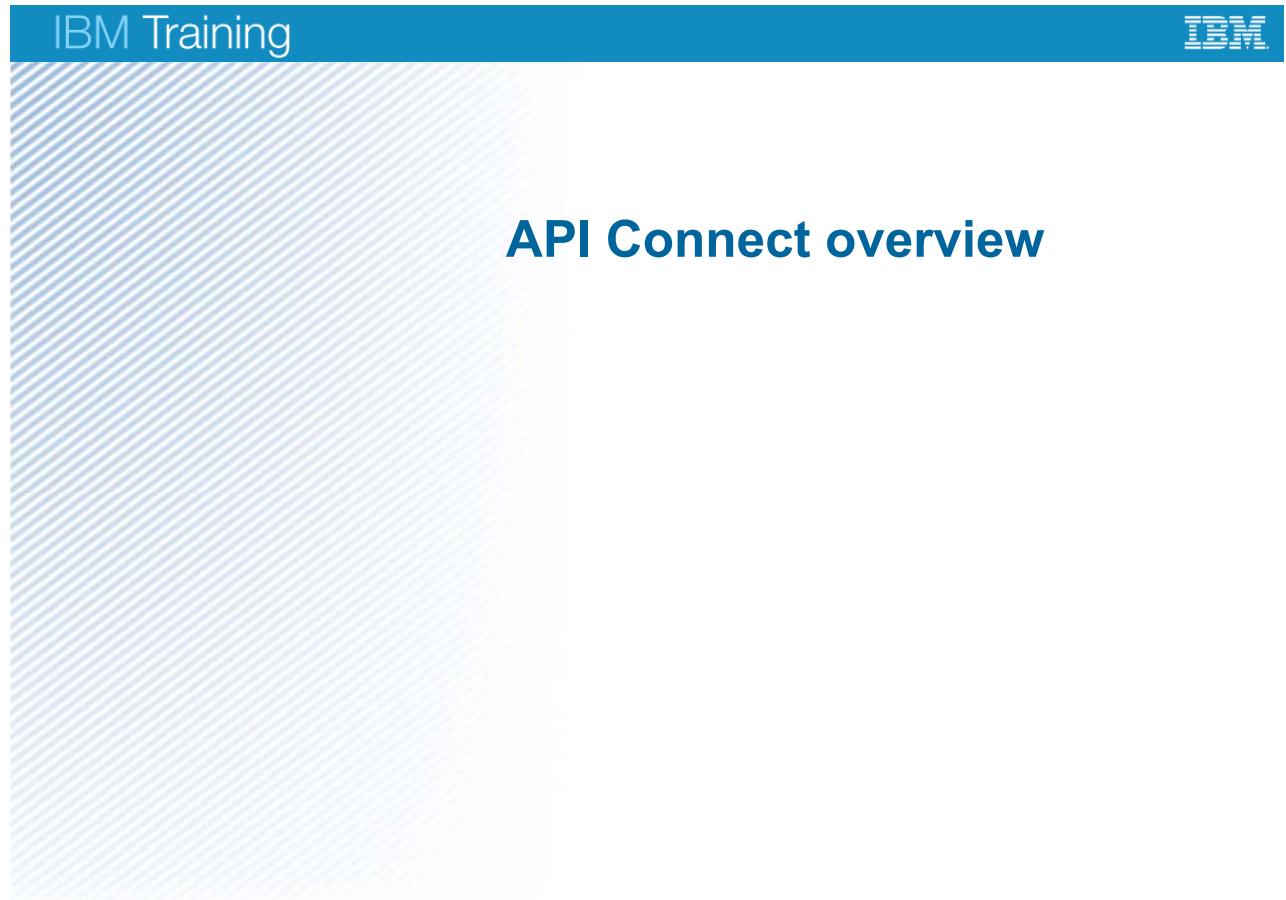
- Explain the role of IBM API Connect in an enterprise architecture
- Identify the components in the API Connect solution
- Identify the deployment environments for API Connect
- Identify the API create, run, manage, and secure features

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-2. Unit objectives*

## 1.1. API Connect overview



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-3. API Connect overview*

## Topics

### API Connect overview

- Gateway options
- Software and hardware prerequisites

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-4. Topics*

## What is API Connect?

- An integrated solution that includes creating, running, managing, and securing APIs for a range of applications in a digital ecosystem
- What does API Connect provide?
  - Automated, visual, and coding options for creating APIs
  - Automated discovery of system of records APIs
  - Node.js Loopback framework support for creating API implementations
  - 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

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

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 for digital applications: mobile, web, Internet of Things, Business Partner applications, and internal applications.

In the following slides, you explore each of the roles and features of IBM API Connect.

## API Connect: Simplified and comprehensive API foundation



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

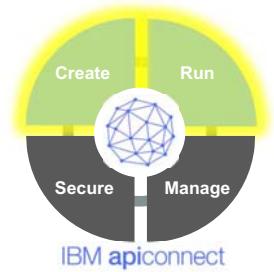
Figure 1-6. API Connect: Simplified and comprehensive API foundation

IBM API Connect is a simplified and comprehensive API solution that focuses on all stages of an API lifecycle: API creation, service runtime administration, API security, and lifecycle management.



## Create and run APIs

- Write APIs in Node.js, Java, or Swift
- 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 bottom-up or top-down
  - Automatic generation of API models
- Local API development and testing
- Visual API composition with the API Designer
- Runtime management that uses a choice of container orchestration software
  - Docker swarm, Kubernetes



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

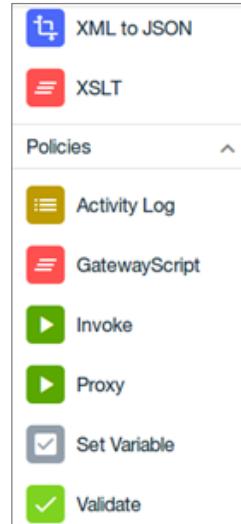
*Figure 1-7. Create and run APIs*

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. You can visually compose message processing policy flows in the API Designer editor. To test your policies and APIs, you can deploy to a local gateway and application, or stage and publish the API, plans, and products to your API Connect Cloud.

You can deploy your application on an open source container software platform, such as Docker. A developer uses containers to package an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. Current versions of Docker include swarm mode, a Docker-native clustering system.

## 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 JavaScript and XSLT



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*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 limiting, handles content-based routing, and transformation between message formats and transport protocols.



IBM

## 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 V5 overview

© Copyright IBM Corporation 2017

*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 available APIs, view details, test, register applications, generate keys, and view analytics on usage
- Social portal with blogs, forums, and ratings



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

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. Application Developers can view public APIs and can sign on to the Developer Portal to see other restricted APIs for which they are granted access.

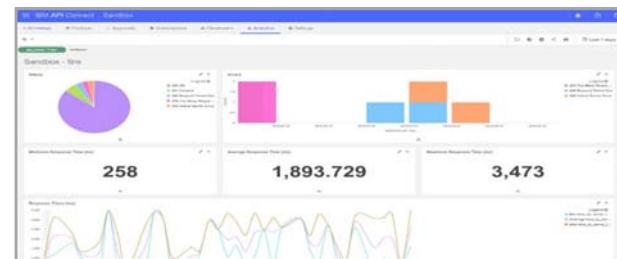
The Developer Portal is based on Drupal, which is a free and open source platform that you can use to manage the content of your website. The Developer Portal that is part of the API Connect software includes blogs, forums, and ratings for APIs as part of the standard offering.

Developers use the self-subscription page to discover and register their client applications without interaction from the system administrator. Your developers can use a set of developer community tools to share information. Application developers can browse available APIs, view details, test, register applications, generate 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
- Enables chargeback for billing API usage
- Easily offload analytics to popular systems like Splunk



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-11. Analyze APIs*

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

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

API analytics in API Connect V5 is built on the Kibana open source analytics and visualization platform, which is designed to work with the Elasticsearch real-time distributed search and analytics engine.

The Elasticsearch engine does logging, indexing, and analysis of log and metric data.

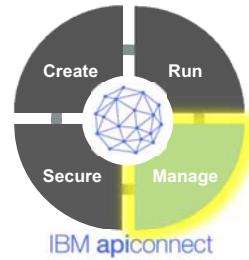
Kibana provides the user interface for visualizations.

You 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 Cloud Manager console and the API Manager user interface. Consumer API analytics can be viewed from the Developer Portal.



## Monetize APIs

- Intuitive user experience for setting pricing for API Products
  - When the monetization feature is enabled in API Connect, UI options (to configure billing and set pricing) become visible
- Stripe integration for easy payments
- API providers can set the pricing details for their products in API Manager or with the API Designer
  - Each plan can be assigned a price
- App developers can easily register on the Developer Portal
- Enter credit card details when subscribing to a paid plan



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

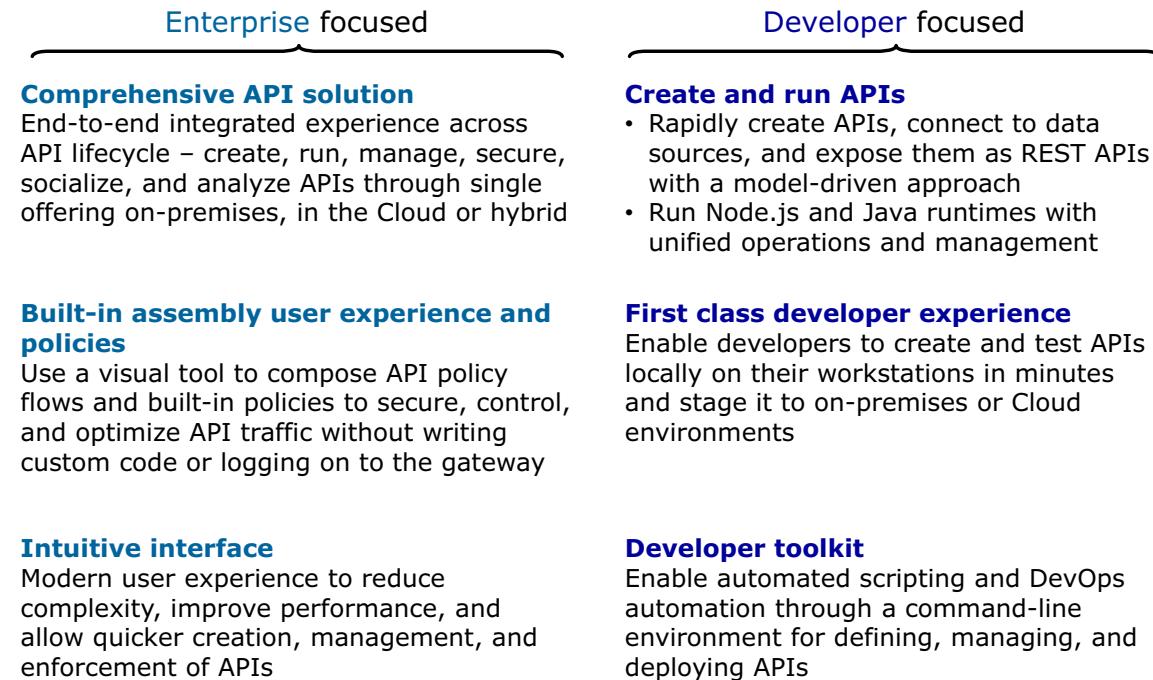
Figure 1-12. Monetize APIs

As soon as the monetization feature is enabled in API Connect, UI options (to configure billing and set pricing) become visible. API providers can then set the pricing details for their products. App developers can easily register on the Developer Portal, and can enter credit card details when subscribing to a paid plan.

The complexity of hosting a payment collection page is offloaded to Stripe, which is a popular payments engine. Consumer data is securely sent to the Stripe server, and nothing gets saved in API Connect. You can set up charging for API use when the monetization feature is enabled in API Connect. Integrated billing and payment management for your APIs is available in API Connect V5.0.8. A Stripe account or Stripe test account is required to process credit card payments.

To set up integrated billing and payment management in API Connect, first add Stripe Integration by using the Admin option in the API Manager user interface.

## Create, run, manage, and secure



IBM API Connect V5 overview

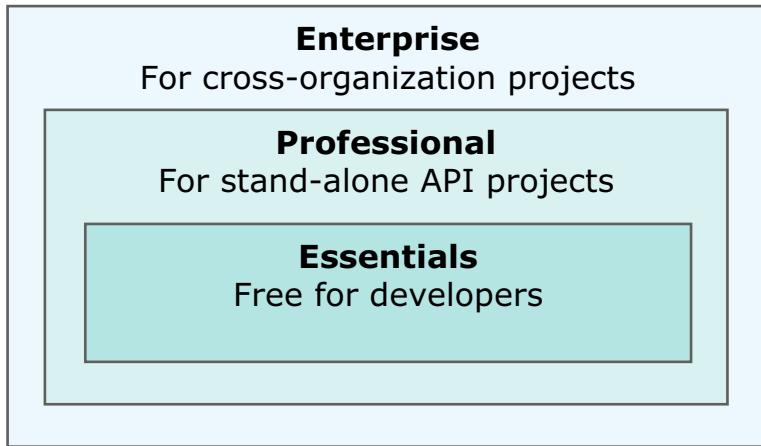
© Copyright IBM Corporation 2017

Figure 1-13. Create, run, manage, and secure

On the left side, the enterprise-focused features offer API lifecycle management with tools to control, secure, catalog, and analyze APIs. On the right side, developer-focused features help developers rapidly create, test, and run APIs. Developers use the API Connect Developer Toolkit to start building, testing, and publishing APIs from their own notebooks.

## API Connect: Grows with your business needs

- Flexible licensing and deployment



Deploy where it is most convenient for you:

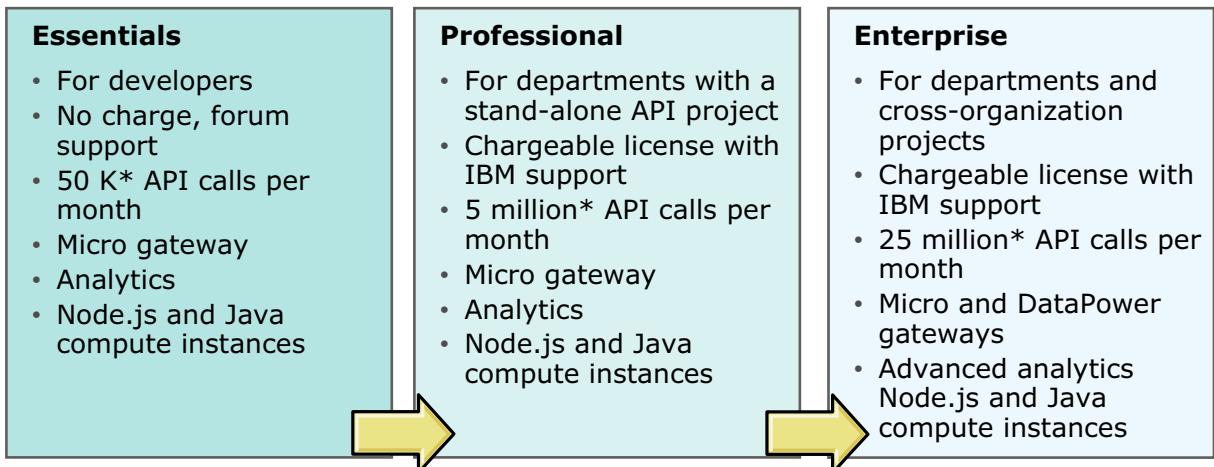
- IBM Cloud
- Third-party Clouds
- On-premises

Figure 1-14. API Connect: Grows with your business needs

The IBM API Connect solution has three entitlement offerings: enterprise, professional, and essentials. You learn about the capabilities of each offering on a later slide.

## Key differences between API Connect offerings

- Licensing: Pay only for what you need
  - Subscription (API calls per month)
  - Perpetual (per PVU, unlimited API calls)



For a complete list of features between API Connect offerings, see:

[www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.overview.doc/overview\\_rapic\\_offerings.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/overview_rapic_offerings.html)

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-15. Key differences between API Connect offerings

Flexible licensing: Pay for only what you need:

- Subscription: API calls per month
- Perpetual: IBM Processor Value Unit, unlimited API calls

\* For more information about the most up-to-date API call entitlements and feature list for each offering level, see the following website:

[https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.overview.doc/overview\\_rapic\\_offerings.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/overview_rapic_offerings.html)

This product provides consumption-based subscription options for customer-managed and IBM-managed offerings.

- The **Essentials** level is for developers. It is available at no charge, but provides only forum-based support.
  - You are allowed 50,000 API calls per month.
  - It provides the Micro Gateway API gateway.
  - It supports analytics, Node.js, and Java compute instances.
- The **Professional entitlement** level is designed for departments with a stand-alone API project.

- It has a chargeable license with IBM support.
- It supports 5 million API calls per month.
- By default, it provides the Micro gateway.
- It also provides Node.js and Java compute instances.
- The **Enterprise entitlement** level is for departments and cross-organization projects.
  - It is a chargeable license with IBM support.
  - It supports 25 million API calls per month.
  - It provides both the Micro gateway and the DataPower-based gateway.
  - It supports advanced analytics and Node.js and Java compute instances.

See the website for the most up-to-date API Connect entitlements for each level.

For more information about the feature list in each offering, see the IBM Knowledge Center article on offering entitlements.

## What is the API Connect Cloud?

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



The **API Management server** 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 1-16. What is the API Connect Cloud?

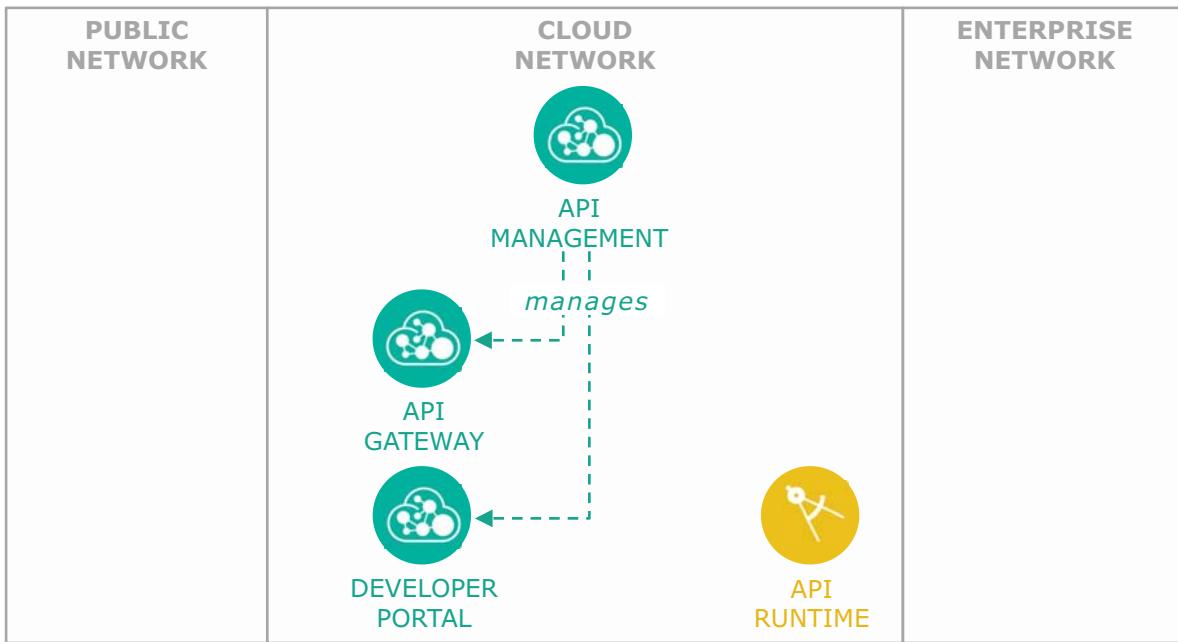
The lab environment for this course consists of four virtual machines: an API Management server, a DataPower API Gateway, a Developer Portal server, and an Ubuntu Linux workstation in which the API Connect Toolkit is installed.

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

- The **API Management server** 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.

## API Connect: Installation and configuration

- The **API Management** server maintains the configuration of the servers in an API Connect installation



IBM API Connect V5 overview

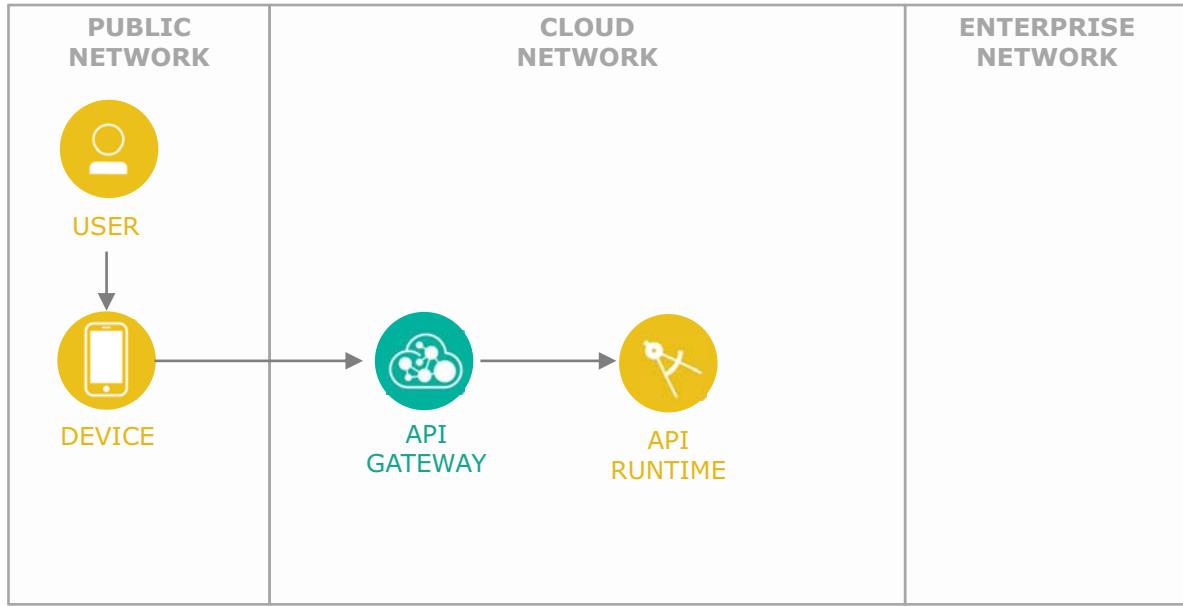
© Copyright IBM Corporation 2017

Figure 1-17. API Connect: Installation and configuration

The **API Management server** manages the configuration of an API Connect installation. The Management server provides two administration interfaces: the **Cloud Management Console** and the **API Manager**. You must register each Developer Portal, API gateway, and API runtime environment in the **Cloud Management Console**. You also manage the lifecycle of staged APIs, plans, and products in the **API Manager** web application.

## API Connect: Runtime architecture (1 of 2)

- The **API gateway** is the **security and policy enforcement point** for all API requests
  - It forwards requests to the application that implements API operations



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

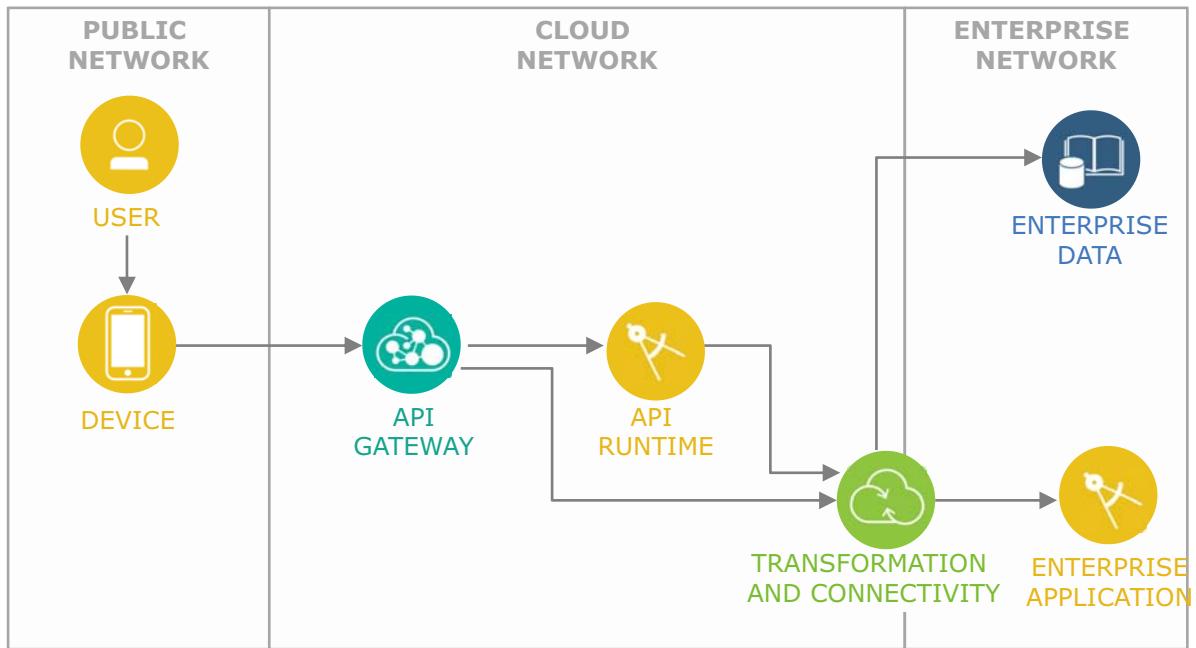
Figure 1-18. API Connect: Runtime architecture (1 of 2)

- The **user** opens an application that calls operations from a published API.
- The computing **device** runs a client application that calls an API operation. The device can be a workstation computer, a mobile device, or an application server on the network.
- The **Cloud network** provides the Cloud platform that hosts the API Connect solution. The Cloud network can be a number of virtual machines that are configured for an on-premises private Cloud, or a public Cloud such as IBM Cloud.
- The **API gateway** is one of four components in an API Connect installation. The gateway validates and load balances API requests.
- In turn, the API gateway calls the application that implements the API operation in an **API runtime** environment. You can deploy the API application in a container, such as Docker or Kubernetes, and can deploy these containers in a private Cloud or a public Cloud.

The **API gateway** is the **security and policy enforcement point** for all API requests. It forwards requests to the application that implements API operations.

## API Connect: Runtime architecture (2 of 2)

- The **API gateway** can also retrieve information from **data sources** and **applications** in your **enterprise architecture**



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-19. API Connect: Runtime architecture (2 of 2)

Many companies have existing applications and databases that do not support direct API access. The **API gateway** can retrieve information from these resources in the enterprise network directly by using REST or SOAP calls to these services. These components can transform the data format and communication protocol when they communicate with data sources and enterprise applications.

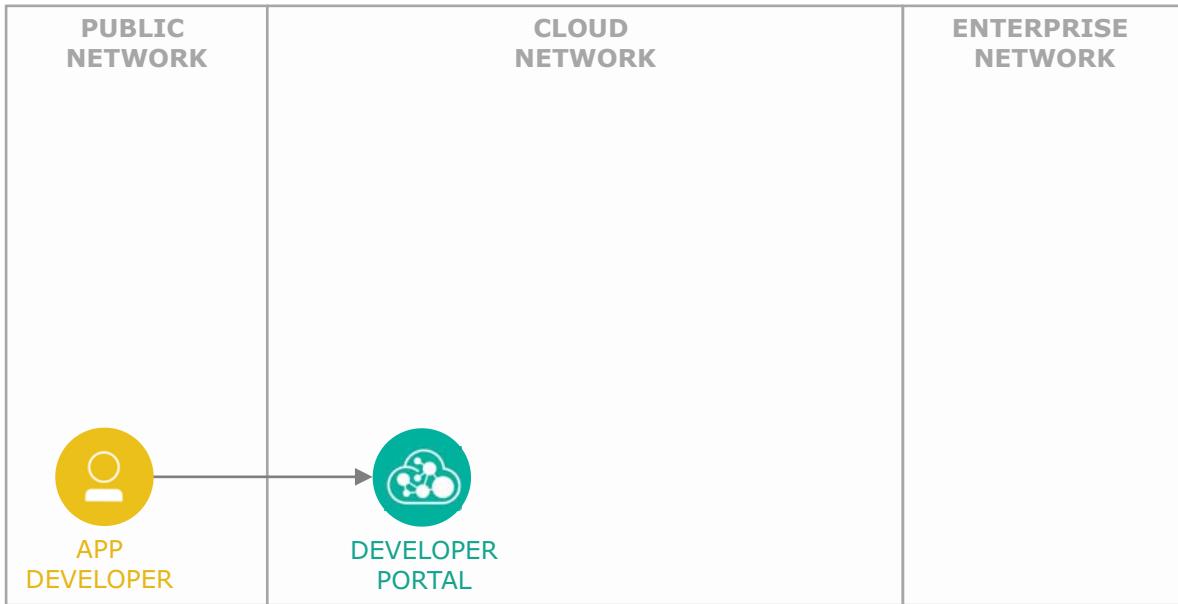
For more complex API implementations, the application that implements the API can also call remote services that run on the API runtime environment and access the data sources by using database connectors.

The **API gateway** can also retrieve information from data sources and applications in your enterprise architecture.

Many companies have existing applications and databases that do not support API access. The API gateway can retrieve information from these resources in the enterprise network through adapters and connectors. These components transform the data format and communication protocol to communicate with data sources and enterprise applications.

## API Connect: Developer Portal

- The **Developer Portal** maintains a catalog of published APIs and the applications that call the API



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

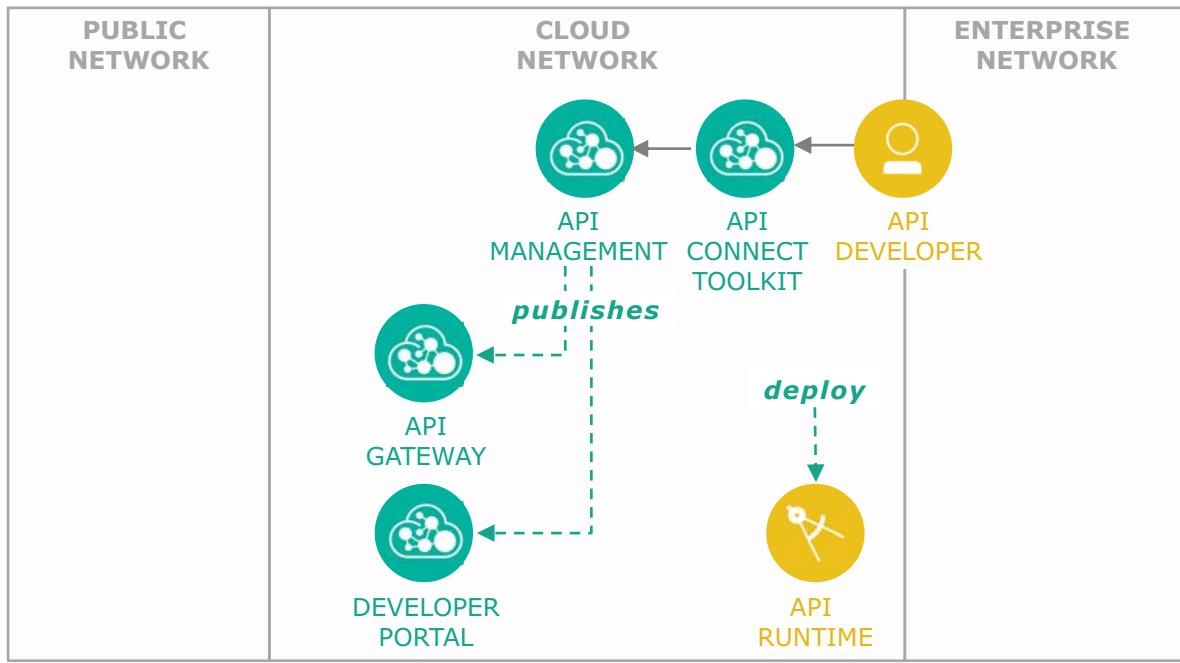
Figure 1-20. API Connect: Developer Portal

The **app developer** is a software developer who creates API clients: applications that call APIs at the API gateway. The app developer must register for an account in the Developer Portal, and subscribe to one or more published services.

The **Developer Portal** supports self-registration for app developers, making it easy to browse, test, and subscribe to APIs. In this diagram, the app developer is a user that logs in to the Developer Portal in the Cloud network.

## API Connect: Development architecture

- API developers create, test, and publish APIs with the **API Connect Toolkit**



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

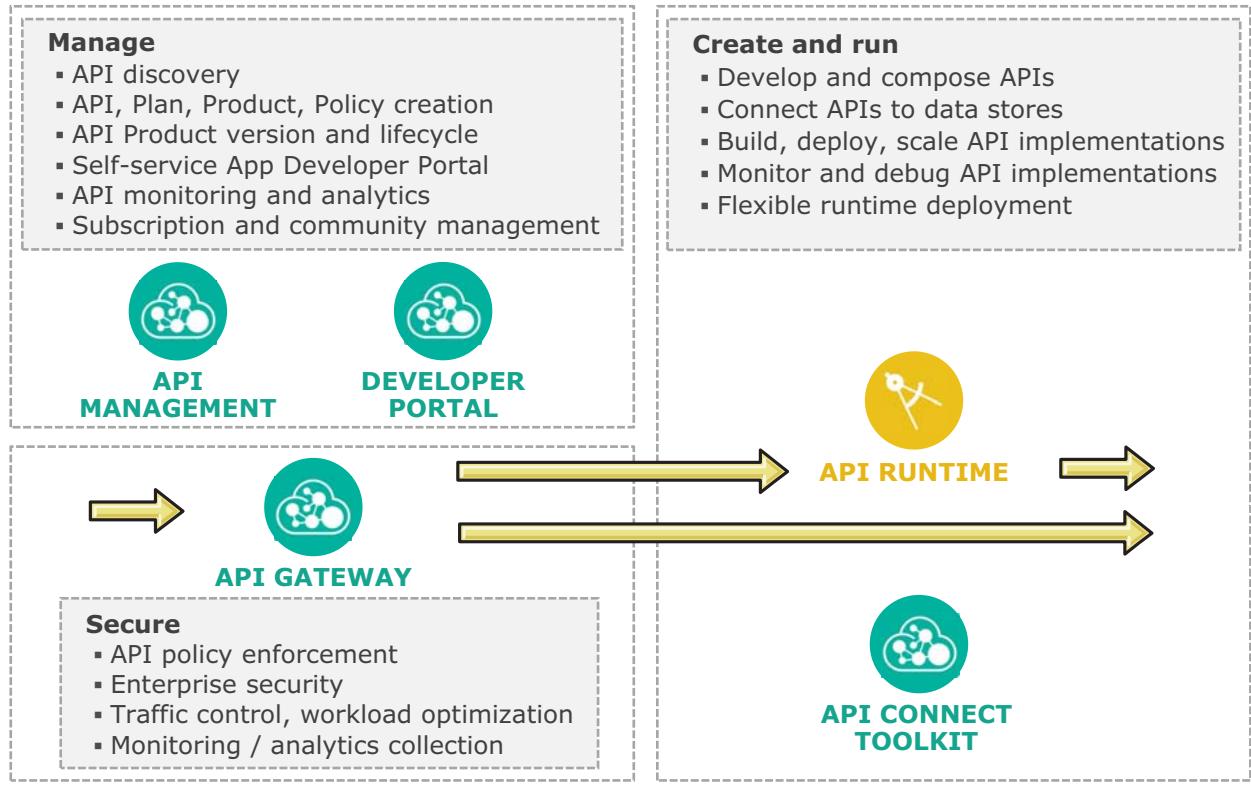
Figure 1-21. API Connect: Development architecture

This course focuses on the role of the API developer. You create, secure, test, and publish APIs to the API Connect Cloud. You use the API Designer web application and “apic” command-line utility from the **API Connect Developer Toolkit** to do these tasks.

After you complete your local testing, you publish the API, plans, and products to the **API** Management server. The publish action publishes the API definition to the API gateway, and it publishes API, plans, and product details to the Developer Portal.

You deploy the API application and its runtime prerequisites to a containerized runtime, such as Docker, by using the container-specific build commands.

## API Connect: Components by feature



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-22. API Connect: Components by feature

The components in API Connect fulfill one or more features in API Connect: API Creation, API Runtime, API Management, and API Security.

The **API Management** and **Developer Portal** servers support the manage features.

The **API Gateway** server supports the secure features.

The **API Runtime** and **API Connect Toolkit** components support the create and run features.

# IBM Training



## Components and roles

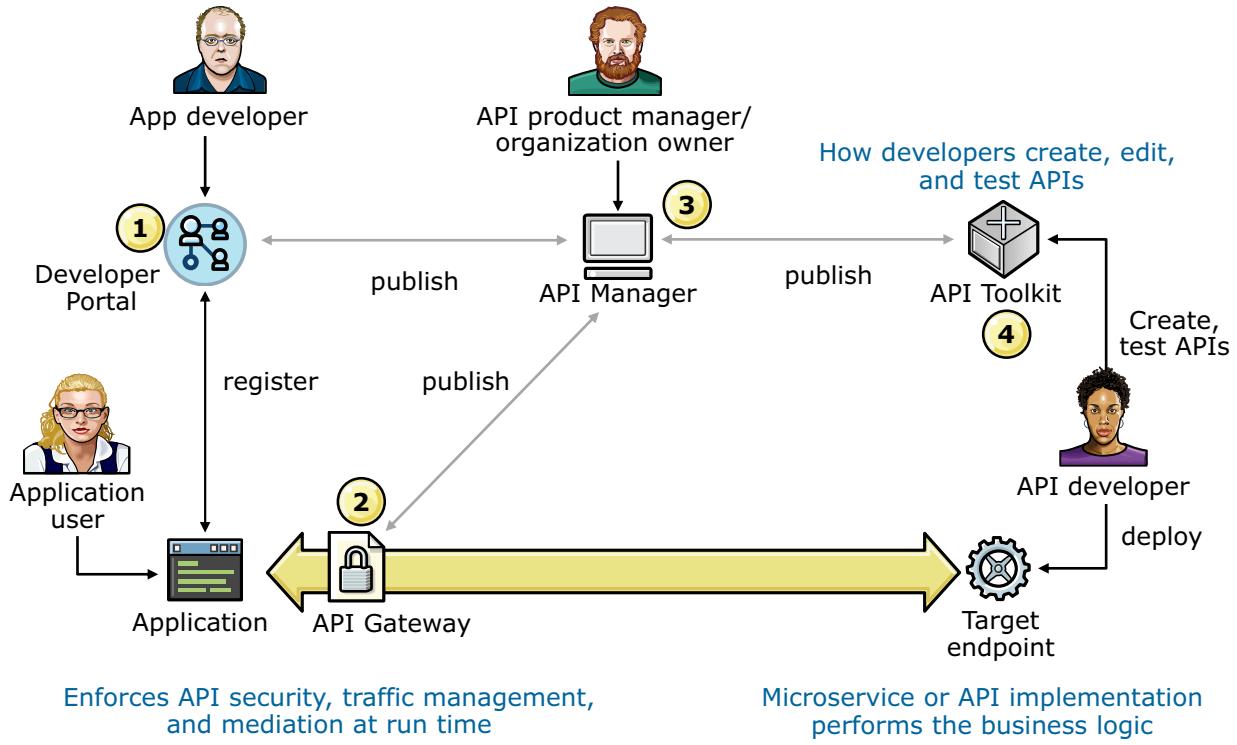


Figure 1-23. Components and roles

The components of the API solution are as follows:

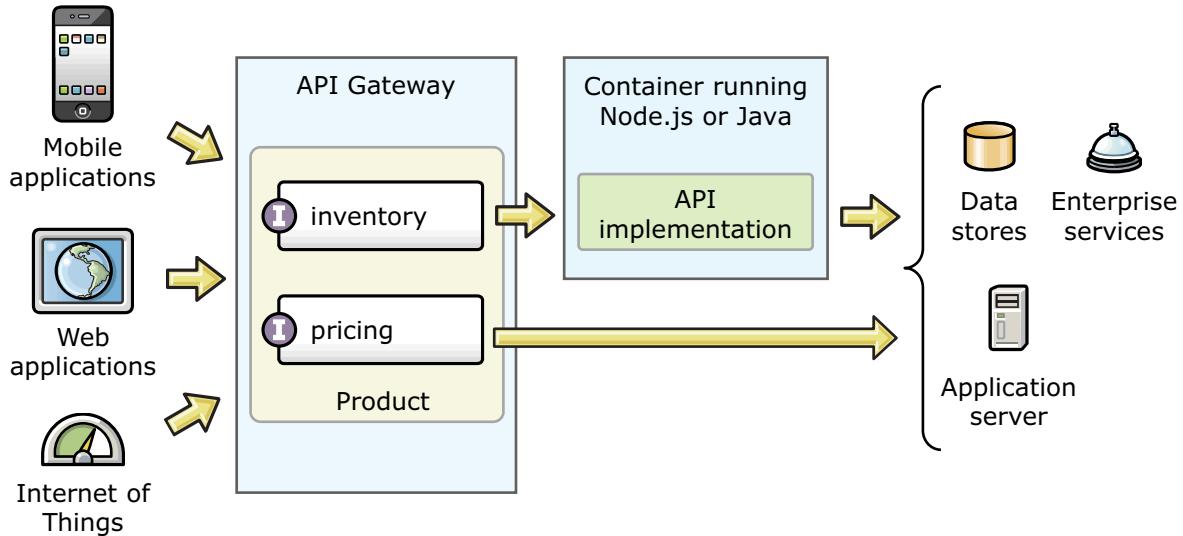
1. The Developer Portal is used to socialize the APIs, register applications, and subscribe to plans.
2. The API Gateway enforces API security, traffic management, and mediation at run time.
3. The API Manager controls the management and publishing of APIs, plans, and products.
4. The API Developer Toolkit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

## API runtime message flow



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-24. API runtime message flow

In this network diagram, you have a number of different client types that want to consume 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.

The **product** is a collection of APIs that are organized together. In your system of engagement, you can publish many products, and different versions of products.

## 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 1-25. 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 must 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
  - Use other languages and frameworks, such as **Swift** Kitura
- 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

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-26. Scenario 2: Implement APIs

One of the main drivers for building APIs is to support the demands from your customers. 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 or Swift.

## 1.2. Gateway options

## Gateway options

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-27. Gateway options*

## Topics

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

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-28. Topics*

## API Gateway options

- The **API Gateway** enforces security, traffic, and message processing policies for your API operations
- In an API Connect solution, you have two implementation choices for the API Gateway: **Micro gateway** and **DataPower** gateway
- API Connect **Enterprise** includes **DataPower** Gateway virtual edition, which provides a secure, performant, field-proven API gateway
  - This option is the upgrade path for existing IBM API Management V4 clients
- API Connect **Professional** and **Essentials** include a programmable **Micro Gateway**, which provides basic functions to empower developers and supports single department projects that are starting on their API journey
  - Option to upgrade to a DataPower Gateway to meet advanced, enterprise-grade API Gateway needs

[IBM API Connect V5 overview](#)

© Copyright IBM Corporation 2017

*Figure 1-29. API gateway options*

An example of an enterprise-grade API gateway need is OAuth 2.0 authentication support. Only the DataPower supports this feature in IBM API Connect.

## Gateway features

- DataPower gateway
  - Built for departments and cross-enterprise use
  - Enterprise-grade security, performance, and stability
  - Low touch gateway without external dependencies (physical, virtual, Cloud, Docker form factors)
  - Comprehensive set of security, traffic management, mediation, and acceleration functions
  - Supports multiple catalogs per instance or cluster
  
- Micro gateway
  - Built for developers and single API projects
  - Programmable with JavaScript, built on Node.js
  - Embedded into the Developer Toolkit experience
  - Basic set of security and traffic management features
  - Supports a single catalog per instance or cluster

*Figure 1-30. Gateway features*

This slide lists the main features of each API gateway type. The following slide compares and contrasts the capabilities between the DataPower gateway and the Micro gateway.

The main features are as follows:

- The DataPower gateway, which is built for departments and cross-enterprise use
- Enterprise-grade security, performance, and stability
- Low-touch gateway without external dependencies (and available in physical, virtual, Cloud, Docker form factors)
- Comprehensive set of security, traffic management, mediation, and acceleration functions
- Supports multiple catalogs per instance or cluster
- The Micro gateway, which is built for developers and single API projects
- Programmable with JavaScript, built on Node.js
- Embedded into the Developer Toolkit experience
- Basic set of security and traffic management features
- Supports a single catalog per instance or cluster

## Gateway capabilities (1 of 2)

Capability	DataPower Gateway	Micro Gateway
Built for	Departments and cross-enterprise	Developers, single API project
Deployment form factor	Physical, virtual, Cloud, Docker	Node package manager
Programmable gateway	Limited (1)	Yes
Embedded in Developer Toolkit experience	Yes (2)	Yes
Purpose-built, DMZ-ready, secure gateway platform	Yes (3)	No
Multi-channel gateway: mobile, web, API, B2B, SOA	Yes	No
Enterprise-grade performance	Yes (4)	No
Reverse proxy, SSL/TLS termination	Yes	No (5)
Built-in policies	Advanced (6)	Standard
Rate limit	Yes	Yes
Any-to-any message transformation	Advanced	Yes (7)
Database connectivity, IBM System z connectivity	Yes	Yes (8)

[IBM API Connect V5 overview](#)

© Copyright IBM Corporation 2017

Figure 1-31. Gateway capabilities (1 of 2)

1. For security, you can implement security logic with GatewayScript, XSLT, XQuery, and JSONiq.
2. If Docker is installed on the same workstation as the Developer Toolkit, an embedded DataPower gateway is made available for testing purposes.
3. DataPower provides drop-in deployment and management. Deploying DataPower appliances into the DMZ is an IBM suggested practice.
4. DataPower provides XML and JSON processing at wire speed, and fast hardware-based encryption.
5. Reverse proxy and SSL/TLS termination depends on an external load balancer and IBM HTTP Server.
6. DataPower provides an advanced set of policies in addition to the standard set in the micro gateway.
7. Any-to-any message transformation is possible on the micro gateway with custom Node.js code. However, this practice is not recommended.
8. The micro gateway depends on custom code and connectors for database connectivity. The DataPower gateway supports SQL and IMS connections.
9. The micro gateway requires custom code and connectors for identity and access management system integration. DataPower gateways support standards as services.

Other important concepts are as follows:

- The DataPower gateway is built for departments and cross-enterprise solutions.
- The Micro gateway is built for developers and single API projects.
- The DataPower gateway is available in physical, virtual, Cloud, and Docker deployment form factors.
- The Micro gateway is available by the node package manager (NPM).
- As a programmable gateway, the options for DataPower gateway are limited.
- For security, you can implement security logic with GatewayScript, XSLT, XQuery, and JSONiq.
- The Micro gateway is fully programmable with Node.js.
- The DataPower gateway is embedded with the Developer Toolkit experience (from V5.0.7.2 onwards).
- The Micro gateway is embedded with the Developer Toolkit experience.
- The DataPower gateway is a purpose-built, DMZ-ready, secure gateway platform.
- DataPower provides drop-in deployment and management. Deploying DataPower appliances into a DMZ is an IBM suggested practice.
- The DataPower gateway supports multi-channels, such as mobile, web, API, B2B, and SOA solutions.
- The DataPower gateway is an enterprise-grade performance hardware. DataPower provides XML and JSON processing at wire speed and fast hardware-based encryption.
- The DataPower gateway supports reverse proxy and SSL/TLS termination.
- The Micro gateway does not support this feature, but you can implement it on your own with third-party Node.js packages.
- The DataPower gateway includes an advanced set of built-in message processing policies. The Micro gateway has only a subset of the standard message processing policies.
- Both the DataPower gateway and the Micro gateway support rate limiting.
- The DataPower gateway supports any-to-any message transformations.
- With the Micro gateway, you must write your own code to implement this solution.
- The DataPower gateway has database connectivity and IBM System z connectivity.
- In the Micro gateway, you must add more third-party packages.

## Gateway capabilities (2 of 2)

Capability	DataPower Gateway	Micro Gateway
Identity and Access Management system integration	Advanced	Yes (9)
Built-in JSON to SOAP transformation	Yes	No
Built-in threat protection (JSON, XML)	Yes	No
Built-in advanced authentication, authorization, and security token conversion	Yes (10)	No
Built-in message encryption, digital signature, compression	Yes	No
Built-in response caching	Yes	No
Front-end self-balancing, back-end load balancing	Yes	No
Non-HTTP protocol and transport protocol conversion	Yes (11)	Yes
Integration with IBM Security Access Manager, MobileFirst, WebSphere Service Registry and Repository	Yes	No
Integration with network hardware security module, anti-virus scanners	Yes (12)	No

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-32. Gateway capabilities (2 of 2)

10. DataPower gateways support WS-Security, LTPA tokens, Kerberos, SAML assertions, and others.
11. The micro gateway supports non-HTTP protocol through custom code. The DataPower gateway natively supports IBM MQ, EMS, Java Message Service, and IMS protocols.
12. DataPower gateways support Gemalto hardware security modules, and ICAP for antivirus.

Other important concepts are as follows:

- The DataPower gateway supports built-in JSON-to-SOAP transformation.
- The Micro gateway does not.
- The DataPower gateway has built-in threat protection from JSON and XML-based attacks.
- The DataPower gateway has built-in advanced authentication, authorization, and security token conversion. For example, the DataPower gateway supports WS-Security, LTPA tokens, Kerberos, SAML assertions, and others.
- The DataPower gateway has built-in message encryption, digital signature, and message compression. It also has built-in response caching, front-end self-balancing, and back-end load balancing.

- The DataPower gateway supports non-HTTP protocol and transport protocol conversion.
- The Micro gateway supports non-HTTP protocol through custom code.
- The DataPower gateway natively supports IBM MQ, EMS, Java Message Service, and IMS protocols.
- The DataPower gateway has integration with IBM Security Access Manager, MobileFirst, WebSphere Service Registry and Repository.
- Last, the DataPower gateway has integration with the network hardware security module and anti-virus scanners. DataPower gateways support Gemalto hardware security modules and ICAP for antivirus.

## What is wrong with this picture?

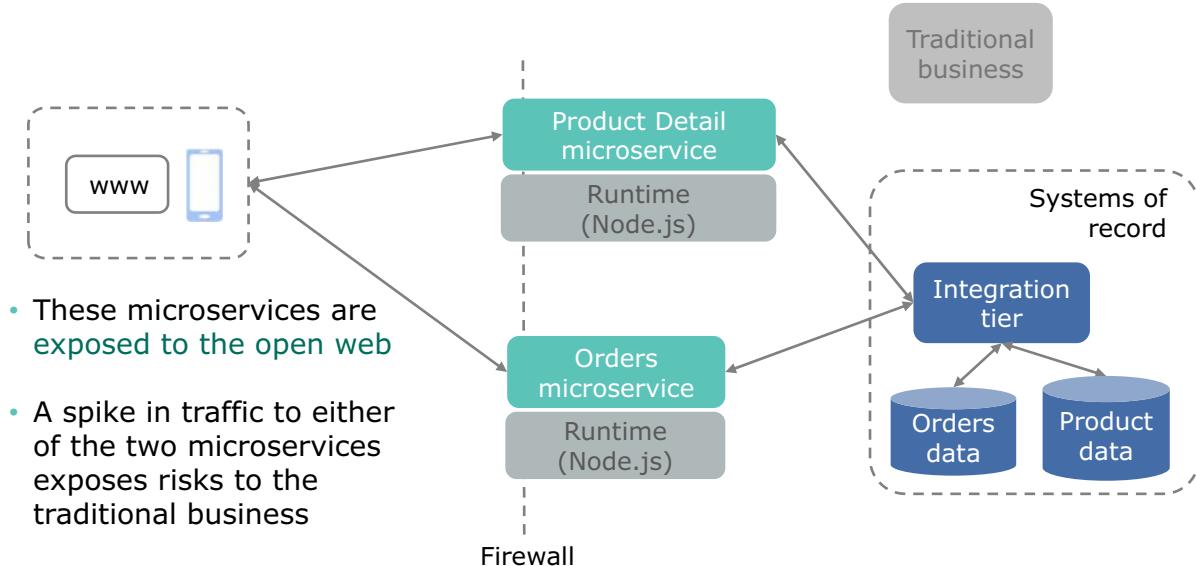
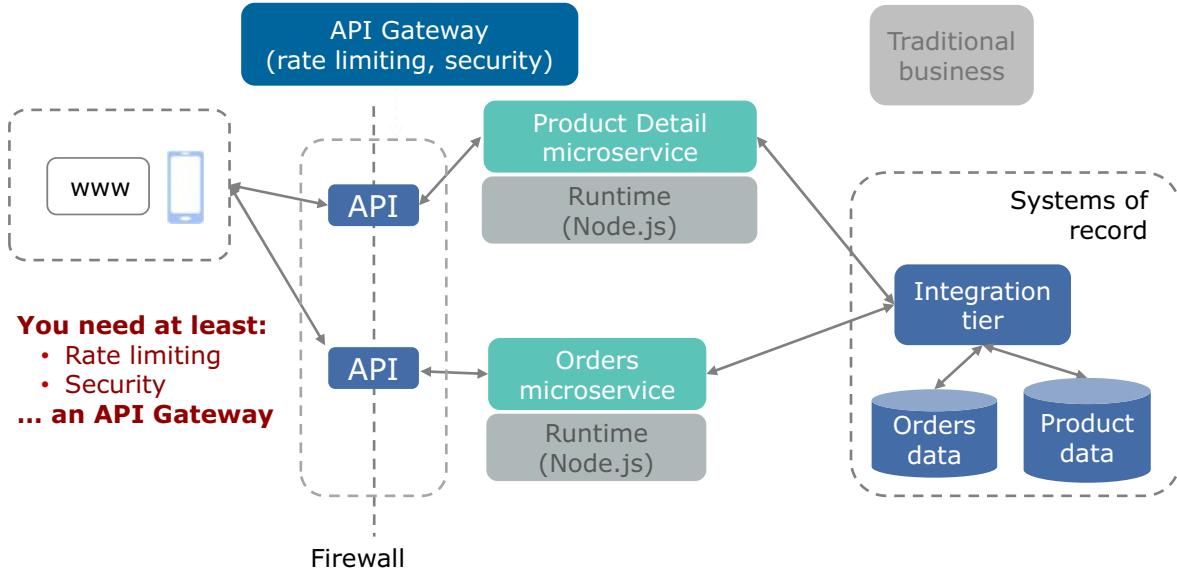


Figure 1-33. What is wrong with this picture?

In this picture, you see that the system of record is shared between the digital business (the two microservices that service the mobile and web apps) and the traditional business. In this picture, nothing controls how much traffic goes through the Product Detail microservice. A spike in the volume of traffic to either of the two microservices can bring down the microservices themselves and potentially the systems of record, which in turn would bring down the traditional business. The outage can become costly for the organization.

## You need an API gateway



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-34. You need an API gateway

The solution to the risks that are posed in the previous slide is to use an API gateway. At the minimum, you need rate limiting and security.

The API gateway acts as a gatekeeper for your IT infrastructure, and ensures only authorized access to your services and data.

## 1.3. Software and hardware prerequisites

## Software and hardware prerequisites

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-35. Software and hardware prerequisites*

## Topics

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

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-36. Topics



## API Connect: Requirements

- Review the most recent software and hardware requirements:
  - [https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.overview.doc/overview\\_apimgmt\\_requirements.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/overview_apimgmt_requirements.html)

**IBM API Connect Professional 5.0.8.0**

Detailed System Requirements

Report filters

Available Reports: 5.0.8.0

Utilities: Regenerate Anytime, Download PDF, Print, Provide feedback

Notes: Data a 01:39: Dis New D View

Operating Systems, Hypervisors, Prerequisites, Supported Software, Hardware

Compilers and Languages

Prerequisite: Node.js, Version: v4.4.0 and above, Prerequisite Minimum: v4.4.0 and above, Desktop Component Support: Developer Toolkit

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

IBM API Connect V5 overview © Copyright IBM Corporation 2017

Figure 1-37. API Connect: Requirements

Review the most recent software and hardware requirements from the IBM API Connect Knowledge Center. 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 the role of IBM API Connect in an enterprise architecture
- Identify the components in the API Connect solution
- Identify the deployment environments for API Connect
- Identify the API create, run, manage, and secure features

IBM API Connect V5 overview

© Copyright IBM Corporation 2017

*Figure 1-38. Unit summary*

## Review questions

1. True or False: You can install a version of IBM API Connect at no cost to develop, test, and define 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 but not the micro gateway?
  - A. JavaScript support
  - B. Rate limiting
  - C. Message transformation
  - D. OAuth authentication URL



IBM API Connect V5 overview

© Copyright IBM Corporation 2017

Figure 1-39. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: You can install a version of IBM API Connect at no cost to develop, test, and define 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 but not the micro gateway?

- A. JavaScript support
- B. Rate limiting
- C. Message transformation
- D. OAuth authentication URL

The answer is D.

# Unit 2. API Connect Developer Toolkit

## Estimated time

01:00

## Overview

This unit describes the features of the API development environment, and explains how to use the API Connect Developer Toolkit to design, develop, test, and publish APIs. You learn how to install and verify the installation of the Toolkit on your own workstation.

## How you will check your progress

- Review questions

## Unit objectives

- Explain the purpose of the API Connect Developer Toolkit
- Identify the features of the API Designer
- Identify the features of the apic command-line utility
- Describe the software and hardware requirements of the Developer Toolkit
- Install the Developer Toolkit
- Verify the installation of the Developer Toolkit

## What is the IBM API Connect toolkit?

- You create **API definitions** and **LoopBack** applications with the **IBM API Connect developer toolkit**
- The **toolkit** is a set of applications that you install on your own workstation:
  - You define API interfaces and LoopBack application components with the **API Designer** web application
  - You review and test APIs with the **API Explorer** web application
  - You generate and configure Loopback applications with the **apic** command-line utility
- The toolkit is distributed at no cost

API Connect Developer Toolkit

© Copyright IBM Corporation 2017

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

The IBM API Connect toolkit is available to download at no cost. API developers install the IBM API Connect toolkit software locally on their own workstation.

An OpenAPI definition is a standard format to describe the operations, paths, and endpoint for an API. A LoopBack application is a REST API implementation that is written in Node.js, a server-side JavaScript runtime. You examine both of these concepts in detail later in this course.

The API Connect toolkit is available for developers on all editions of API Connect: essential, professional, and enterprise. Even if you run the IBM Cloud version of API Connect, you can still install the API Connect toolkit locally on your own workstation.

## How do you install the toolkit?

- **Review the most recent installation instructions:**
  - <https://developer.ibm.com/apiconnect/getting-started/>
- The four 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. Install the apiconnect Node package
  4. Verify the toolkit installation

Figure 2-3. How do you install the toolkit?

Before you continue, open the most recent installation instructions from the IBM Developer site. 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.

You can also install the API Connect toolkit from an API Manager. If your workstation does not have access to the NPM repository, this scenario is useful. Review the API Connect documentation for instructions on how to install the toolkit from an API Manager.

## Step 1: Install and configure the prerequisite software

- Mac OS
  - Install Xcode
  - Accept the Xcode license
- Linux
  - Python (check documentation for specific version)
  - make
  - A C/C++ compiler toolchain (check documentation for specific version)
  - On Debian and Debian-based distributions, run:
 

```
$ apt-get install build-essentials
```
- Windows
  - Microsoft .NET Framework
  - Visual Studio with Visual C++ development tools
  - Windows SDK
  - Python (check documentation for specific version)

*Figure 2-4. Step 1: Install and configure the prerequisite software*

For more information about the specific versions of the software that is listed on this slide, see:  
<https://developer.ibm.com/apiconnect/getting-started/>

In this course, the API Connect toolkit is already installed on the student workstation virtual machine. You do not need to install this software to complete the exercises.

On a Mac operating system, install the Xcode application. After you download and install Xcode, accept the Xcode license.

On a Linux operating system, ensure that you have the correct version of Python.

Check the documentation for a specific version.

Check the version of the make utility.

You also need a C and C++ compiler toolchain.

On Debian and Debian-based distributions such as Ubuntu, run the command:

**apt-get install build-essentials**

On the Windows operating system, ensure that you have the correct version of the Microsoft .NET Framework, the Visual Studio with Visual C++ development tools, the Windows SDK, and Python. Check the documentation for the specific version.

For more information about the specific versions of the software that is listed on this slide, see:  
<https://developer.ibm.com/apiconnect/getting-started/>

In this course, the API Connect toolkit is already installed on the student workstation virtual machine. You do not need to install this software to complete the exercises.

## Review the supported level for Node.js

- Open the **Planning and installing IBM API Connect > IBM API Connect Version 5.0 requirements** page in the IBM Knowledge Center
- Select any one of the three offering levels: essentials, professional, or enterprise
- In the Software Product Compatibility Report, select **Prerequisites > Compilers and Languages**

Compilers and Languages			
Prerequisite	Version	Prerequisite Minimum	Components
IBM SDK for Node.js	V4.4.0.0 and future fix packs	V4.4.0.0	
	V4.4.3.0 and future fix packs	V4.4.3.0	
	V4.4.4.0 and future fix packs	V4.4.4.0	

API Connect Developer Toolkit

© Copyright IBM Corporation 2017

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 review the IBM API Connect requirements page from the “Planning and installing IBM API Connect” topic. All three product offering levels include the API Connect toolkit; select one to open the compatibility report.

Review the “IBM SDK for 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.

## Step 2: Install the Node runtime environment and npm

- Install a supported version of Node.js
  - Review “IBM API Connect Version 5.0 requirements” to find the supported Node.js level for API Connect
- Make sure that the `node` command is in your PATH
  - Open a terminal or command prompt window
  - Type `node -v` to review the Node runtime version
- For MacOS and Linux workstations, verify that you can install **global Node packages** with your user account
- For Windows workstations, update your version of the `npm` utility to version 3
  - Run `npm install -g npm` to install the most recent version of npm as a global package

Figure 2-6. Step 2: Install the Node runtime environment and NPM

The default installation directory for the Node.js runtime environment is `/usr/local` in workstations that are running either Mac OS or Linux. Review the “fixing npm permissions” documentation from the [npmjs.org](http://npmjs.org) website to verify and change directory ownership directory. Do not follow these instructions on a server system. Changing directory permissions is appropriate only on a local development system.

On Windows workstations, run the `node` and `npm` commands in the Windows command shell.

## Step 3: Install the apiconnect Node package

- To install the toolkit from npm, enter the following command:

```
$ npm install -g apiconnect
```

- You can safely ignore warnings from the node-gyp module during the installation process
- To install the toolkit from a Management server in your API Connect Cloud, enter:

```
$ npm install -g --unsafe-perm apiconnect  
https://appliance/packages/apiconnect
```

- The --unsafe-perm parameter runs the installation command with root privileges
- If your account does not have root level access, you must add the unsafe permission parameter

Figure 2-7. Step 3: Install the apiconnect Node package

The API Connect toolkit uses the “npm” package management utility to download and install the application to your local workstation. After you install and configure the prerequisite software on your computer, run the `npm install` command to install “apiconnect” as a global package. If your workstation does not have access to the [npmjs.org](http://npmjs.org) website, you can also install the API Connect toolkit from the Management server. Remember to add the unsafe permissions parameter to the `install` command. Otherwise, the API Connect toolkit installation might fail with a non-root user account.

## Step 4: Verify the toolkit installation

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

```
$ apic -v
```

- Start the API Designer web application

```
$ apic edit
```

Figure 2-8. Step 4: 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.

To open the API Designer web application, run: `apic edit`

## Installation troubleshooting: User permissions

- If your user account does not have the required permissions to create files or directories, you might see the following errors:

```
npm ERR! Error: EACCES, mkdir  
'/usr/local/lib/node_modules/apiconnect'  
...  
npm ERR! Please try running this command again as  
root/Administrator  
...
```

- On your local Mac or Linux workstation, you can fix the directory rights in your local user space directory:

```
$ sudo chown -R $USER /usr/local/{lib/node_modules,bin,share}
```

- Warning: Do not run this command on a server system
- Also, do not run this command on the /usr/bin directory

Figure 2-9. Installation troubleshooting: User permissions

If your user account does not have the required permissions to create files or directories, you might see an “EACCES” error from the “npm” utility.

On your local Mac or Linux workstation, you can fix the directory permissions in your local user space directory. However, do not run this command on a server system, or on the /usr/bin directory.

## Installation troubleshooting: Xcode

- On Mac OS, if you see Xcode messages in your installation:

Agreeing to the Xcode/iOS license requires admin privileges,  
please re-run as root via sudo

- Make sure that you validate your Xcode license before you install the “apiconnect” package

```
$ sudo xcode-select
```

Figure 2-10. Installation troubleshooting: Xcode

On MacOS workstations, you must install Xcode and accept the license before you install the toolkit. If you see an error message in your terminal that refers to Xcode, run the `sudo xcode-select` command to start the Xcode application. Accept the license terms, and run the `apiconnect` installation command again.

You can also run the Xcode application from the Finder and accept the license agreement.

## Installation troubleshooting: sqlite

- On Ubuntu Linux, if you see sqlite errors in the console:

```
sqlite3@3.1.1 install /usr/local/lib/node_modules/strong-
pm/node_modules/minkelite/node_modules/sqlite3
node-pre-gyp install --fallback-to-build
/usr/bin/env: node: No such file or directory
npm WARN This failure might be due to the use of legacy binary
"node"
npm WARN For further explanations, please read
/usr/share/doc/nodejs/README.Debian
npm ERR! weird error 127
npm ERR! not ok code 0
```

- Enter the following command:

```
$ update-alternatives --install /usr/bin/node node
/usr/bin/nodejs 99
```

*Figure 2-11. Installation troubleshooting: sqlite*

On Ubuntu Linux workstations, if you see warning or error messages with the `sqlite` package, run the `update-alternatives` command. The command sets the correct `node` application in your workstation.

The purpose of the `update-alternatives` command is to modify the list of applications in the `/etc/alternatives` directory. The `alternatives` directory holds a list of symbolic links that point to an application in your workstation. For example, the `update-alternatives` command that is listed on the slide sets the default, preferred copy of the `node` application. With this fix, the `sqlite3` package installer finds the correct copy of the `node` application.

## Upgrade or reinstall the Developer Toolkit

- If you upgrade your IBM API Connect Cloud to a fix pack level, upgrade your Developer Toolkit installation also
- To upgrade your API Connect toolkit:
  - Stop all locally running API applications

```
$ apic stop --all
```

- Remove your existing copy of “apiconnect” and clear the npm cache

```
$ npm uninstall -g apiconnect --cache-clear
```

- Reinstall the toolkit

```
$ npm install -g apiconnect
```

- To install a particular version of the toolkit

```
$ npm install -g apiconnect@apic-v5.0.8.1
```

*Figure 2-12. Upgrade or reinstall the Developer Toolkit*

If you upgrade the API Management server, Gateway server, and Developer Portal to an updated fix pack level, you must upgrade your API Connect toolkit also.

First, stop all locally running API applications with the `apic stop -all` command. You might have LoopBack Node applications or the Microgateway running in your workstation. Second, remove the existing copy of `apiconnect` with the `npm` command. Remember to add the `--cache-clear` parameter to remove any locally stored copy of the `apiconnect` package. Last, reinstall the toolkit with the `npm install` command.

## API Connect Toolkit: The apic command

- The **apic** command defines, generates, and configures your API interface definition and API implementation
  - Generate API definitions and gateway configuration in the form of OpenAPI definitions
  - Generate a sample LoopBack application to implement APIs
  - Define LoopBack models, properties, and relationships
  - Update an API definition based on your LoopBack API implementation
  - Publish API definitions, Products, and Plans to API Management Server

```
localuser@ubuntu-base:~$ apic
Usage: apic COMMAND OPTIONS

Options
-h, --help      command usage
-v, --version   toolkit version

Commands (type apic COMMAND -h for additional help):

Creating and validating artifacts
config          manage configuration variables
create          create development artifacts
edit            run the API Designer
validate        validate development artifacts

Creating and testing applications
loopback        create and manage LoopBack applications
microgateway    create Micro Gateway applications
start           start services
stop            stop services
logs            display service logs
props           service properties
services         service management

Publishing to the cloud
login           log in to an IBM API Connect cloud
logout          log out of an IBM API Connect cloud
organizations   manage organizations
catalogs        manage catalogs in an organization
publish         publish products and APIs to a catalog
products        manage products in a catalog
apps            manage provider applications
drafts          manage APIs and products in drafts
```

[API Connect Developer Toolkit](#)

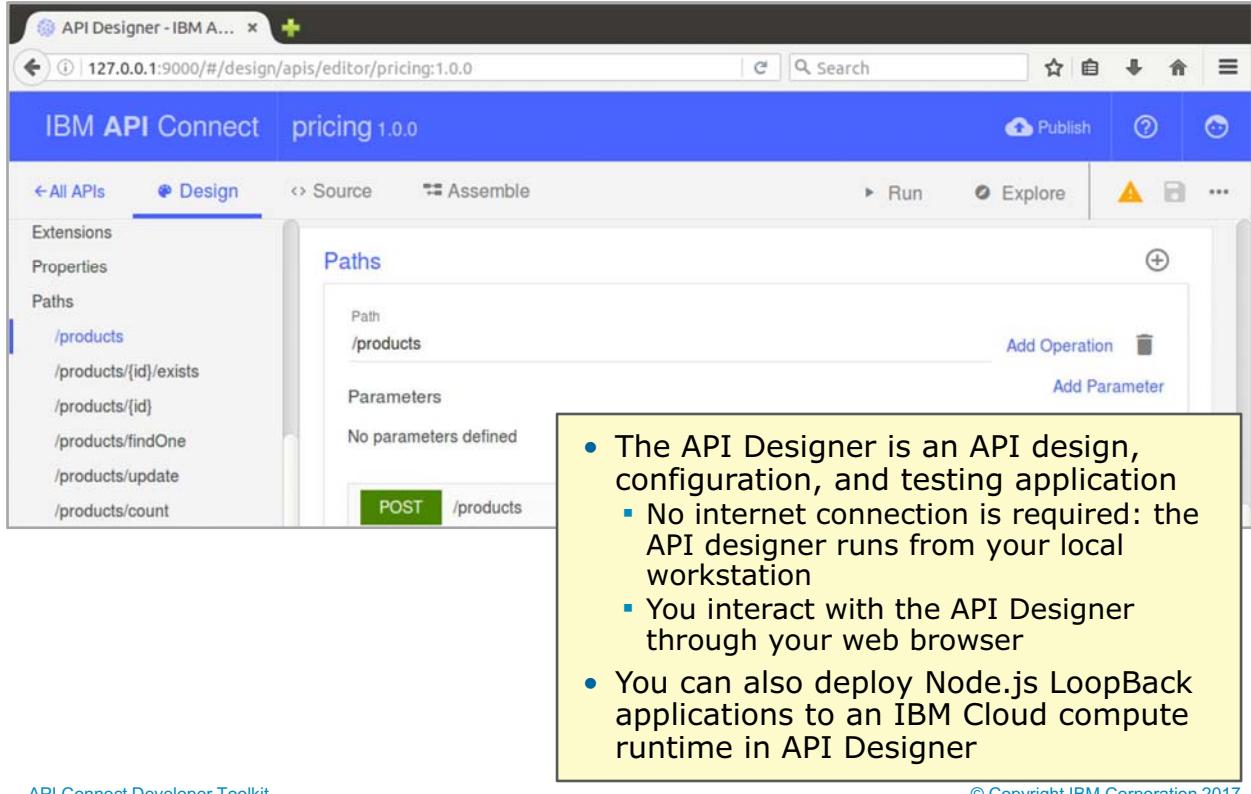
© Copyright IBM Corporation 2017

Figure 2-13. API Connect Toolkit: The apic command

# IBM Training



## API Connect Toolkit: The API Designer web application



API Connect Developer Toolkit

© Copyright IBM Corporation 2017

Figure 2-14. API Connect Toolkit: The API designer web application

The API Designer is a web browser application that your computer hosts. You start the API Designer application with the `apic edit` command.

You can also deploy Node.js LoopBack applications to a Bluemix compute runtime in API Designer.

You start the API Designer application with the `apic edit` command.

## How to create APIs in API Connect

- How do you define an **existing API** in IBM API Connect?
  - Define the API interface and operations in an API definition
  - 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
  - Define the API interface and operations in an API definition
  - Create the models, properties, and relationships in the LoopBack application
  - Implement the business logic in Node.js
  - Deploy the Loopback application to a server
  - Publish the API definition to the API Gateway through API Manager

Figure 2-15. How to create APIs in API Connect

For a detailed discussion and demonstration 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. Strictly speaking, the publish operation takes two steps: you publish the interface of an API as an **API definition** to the **API Manager**. In turn, the API Manager updates the configuration of the **API gateway** with the new API. You examine these steps in greater detail in the upcoming units.

## Unit summary

- Explain the purpose of the API Connect Developer Toolkit
- Identify the features of the API Designer
- Identify the features of the apic command-line utility
- Describe the software and hardware requirements of the Developer Toolkit
- Install the Developer Toolkit
- Verify the installation of the Developer Toolkit

## Review questions

1. True or False: You can download and use the API Connect toolkit at no cost.
2. Which command starts the graphical API Designer application?
  - A. apic design
  - B. apic create
  - C. apic loopback
  - D. apic edit
3. Which tasks can you perform with the “apic” command-line utility?
  - A. Create an OpenAPI definition
  - B. Publish APIs to the API Connect Cloud
  - C. Start and stop API services
  - D. All of the above



API Connect Developer Toolkit

© Copyright IBM Corporation 2017

Figure 2-17. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: You can download and use the API Connect toolkit at no cost.  
The answer is True.
2. Which command starts the graphical API Designer application?
  - A. `apic design`
  - B. `apic create`
  - C. `apic loopback`
  - D. `apic edit`The answer is D.
3. Which tasks can you perform with the “apic” command-line utility?
  - A. Create an OpenAPI definition
  - B. Publish APIs to the API Connect Cloud
  - C. Start and stop API services
  - D. All of the aboveThe answer is D.

API Connect Developer Toolkit

© Copyright IBM Corporation 2017

Figure 2-18. Review answers

## Demonstration: Installing and verifying the Developer Toolkit

API Connect Developer Toolkit

© Copyright IBM Corporation 2017

*Figure 2-19. Demonstration: Installing and verifying the Developer Toolkit*

## Demonstration objectives

- Verify the installation of the Node.js runtime environment
- Install the API Connect Developer Toolkit
- Test the apic command-line utility
- Open the API Designer web application



Figure 2-20. Demonstration objectives

---

# Unit 3. Creating an API definition

## Estimated time

01:15

## Overview

This unit examines the API definition, a structured file that documents the API operations, parameters, and data types. You learn how to document, version, and define your API interface. You also examine the role of environment variables as properties.

## 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
- Document the API version and description
- Define an API operation
- Define query and path parameters
- Define request and response headers
- Explain the purpose of API properties
- Define environment variables as properties
- Modify the target URL as a property
- Explain the concept of API templates

## 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

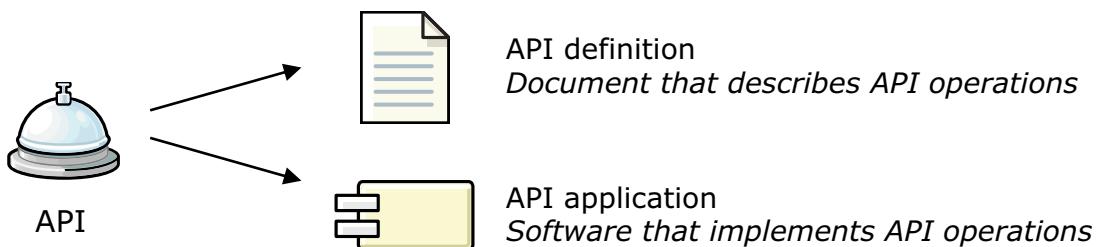


Figure 3-2. What is an API definition?

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 scanning 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. The toolkit also generates documentation from the API definition file.

## 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
- A group of companies, including IBM, founded the Open API Initiative (OAI) to promote and develop an open source standard to define REST APIs
- 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 2017

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

The Open API specification builds on top of the Swagger REST API specification, which is written by SmartBear Software, the creators of the SoapUI test suite. The Open API 2.0 specification is licensed under the Apache License, version 2.0. For more information, see:  
<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>

## OpenAPI definition structure

- Name, description, version
- Contact information, licensing terms
- 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 2017

Figure 3-4. OpenAPI definition structure

This chart outlines the sections in an OpenAPI definition.

The **name**, **description**, **version**, **contact information**, and **licensing terms** entries describe the metadata on the API implementation. These fields are machine readable: the test client in the API Designer Explore view generates online documentation from these fields.

The **host** and **base path** entries define the network endpoint for the API. The API operations, which are defined in the **paths** section, are listed immediately after the base path.

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.

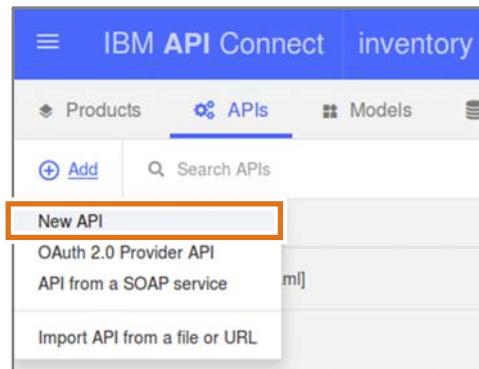
The **properties** store values that are specific to a deployment environment. For example, you can store the server address for the micro gateway in the sandbox development environment.

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. You learn more about these properties later in this unit.

## How do you create an API definition?

- To create an OpenAPI definition, select **Add > New API**
- To start from an existing API definition, select **Add > Import an API from a file or URL**
  
- 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 the IBM API Connect interface. At the top, there's a navigation bar with 'Products', 'APIs' (which is the active tab), and 'Models'. Below the navigation bar is a search bar labeled 'Search APIs' and a 'Add' button. A dropdown menu is open under the 'Add' button, with the 'New API' option highlighted by an orange box. Other options in the dropdown include 'OAuth 2.0 Provider API' and 'API from a SOAP service'. Below the dropdown, there's a link 'Import API from a file or URL'.


The screenshot shows the 'New OpenAPI from scratch' configuration dialog. It has two sections: 'Info' and 'Definition'. In the 'Info' section, there are fields for 'Title' (set to 'savings'), 'Name' (set to 'savings'), 'Base Path' (set to '/savings'), and 'Version' (set to '1.0.0').

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-5. How do you create an API definition?

You have two options to create a REST API definition in API Designer. You can start with an empty OpenAPI definition document with the **New API** option. The **Import API from a file or URL** wizard downloads an OpenAPI document from your workstation or a remote server.

You examine the **OAuth 2.0 Provider API** and **API from a SOAP service** options in a later unit.

## OpenAPI definition: Name, version, and licensing

- The **info** and **contact** entry provides metadata on the published API
- The **title** and **description** describe the name and a short overview of the API application
- The **terms of service** points to a web page with your organization's terms for the API
- The **license** links to a published, legal statement for using the API definition and implementation
- 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

Info	Title * savings  Name * savings  Version * 1.0.0  Description  Savings plan estimator to calculate growth
Contact	Name Thomas Watson  Email thomas@think.ibm.com  URL www.ibm.com

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-6. OpenAPI definition: Name, version, and licensing

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

## Parts of an REST API definition

Type	Description	Example
Host	<ul style="list-style-type: none"> <li>The name or address of the server that hosts the API</li> <li>In API Connect, the <b>host</b> is the address of the API Gateway</li> </ul>	http://api.think.ibm or \$(catalog.host)
Base path	<ul style="list-style-type: none"> <li>The web route that identifies the API</li> <li>It appears immediately after the host name</li> </ul>	/api
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	Sandbox: http://localhost:4001/

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-7. 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.

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 sandbox development environment and the production environment have different hosts. Properties avoid storing hardcoded values at the API gateway or API application.

## Parts of an REST API operation

Type	Description	Example
Path	The web resource, immediately after the base path	/Plans/estimate
Operation	The API operation is an action with an API resource	GET /Plans/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 2017

Figure 3-8. 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: /Plans/estimate.
- The **operation** is the HTTP method that acts on the resource, for example: GET /Plans/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 2017

Figure 3-9. 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.

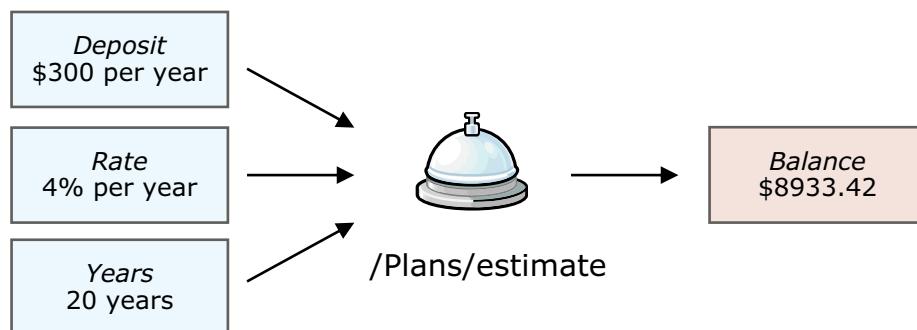
- A **GET** method retrieves the entity that is represented as a resource path on the server. The operation is **safe** because it is read-only: it does not change the information on the server. It is also **idempotent**: calling a GET method multiple times in succession returns the same value.
- The **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 /Plans/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** specifies 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 2017

Figure 3-10. Example: Savings plan estimate

The **savings plan** REST API is a sample application that is hosted on the IBM Bluemix architecture. The API hosts a variant of the **/Plans/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.

Try out the sample API implementation at: <https://savingsample.mybluemix.net>

## Example: GET /Plans/estimate

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

```
GET /api/Plans/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 3-11. Example: GET /Plans/estimate

In this example, you send an HTTP GET request to the **savings plan** API operation that is hosted on Bluemix.net. The REST operation expects input parameters as query parameters: a set of key-value pairs after the question mark (?) character. The HTTP request accepts header indicates that the calling application expects a response with the **application/json** media type.

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

## Step 1: Define a path, and operation

- The **path** represents a resource that is hosted on the server
  - For example, **/Plans** represents savings account plans

- 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 /Plans/estimate** operation, you retrieve data on the savings plan estimate resource

The screenshot shows two panels. The top panel is titled 'Paths' and contains a list with one item: '/Plans/estimate'. The bottom panel is titled 'GET /Plans/estimate' and includes fields for 'Add Tag', 'Summary', 'Operation ID', and 'Description'. The 'Description' field contains the text: 'Calculate savings growth with annual compounding'.

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-12. Step 1: Define a path, and operation

In this example, you want to define an OpenAPI definition that describes the GET /Plans/estimate operation in the Bluemix 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 **/Plans/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 /Plans/estimate** operation retrieves a savings plan estimate calculation.

## Step 2: Define the input parameters and response message

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

Parameters					
Name	Located In	Description	Required	Type	Add Parameter
deposit	Query	Annual deposits	<input checked="" type="checkbox"/>	float	
rate	Query	Interest rate	<input checked="" type="checkbox"/>	float	
years	Query	Years of saving	<input checked="" type="checkbox"/>	integer	

Responses		
Status Code	Description	Schema
200	200 OK	plan-result

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-13. Step 2: Define the input parameters and response message

Enter 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 **plan-result** custom data type in the message body

## Step 3: 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 **plan-result** type defines a JSON object with one property: **balance**
- You specified this custom data type as the return value in the **GET /Plans/estimate** operation

Definitions				
plan-result				
Name *	plan-result			
Type	object			
Description	Calculated result from savings plan			
Edit				
Properties				
*	Property Name	Description	Type	Example
<input type="checkbox"/>	balance	Account balance	float	8933.4

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-14. Step 3: Define the result data type

## Example: POST/Plans/estimate

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

```
POST /api/Plans/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 }
```

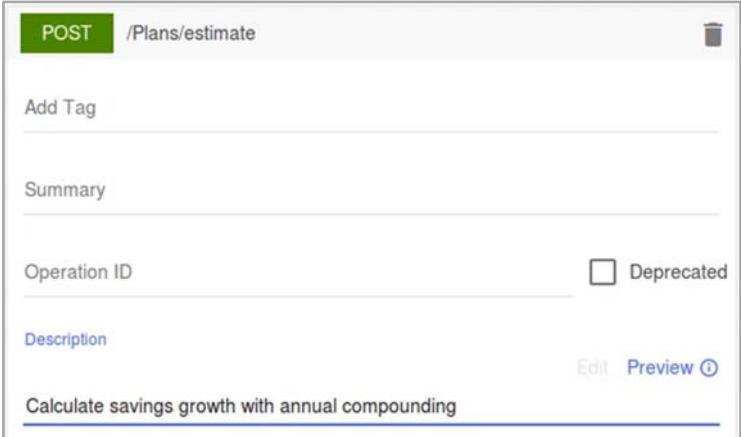
HTTP response message

Figure 3-15. Example: POST/Plans/estimate

The **POST /Plans/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 /Plans/estimate** operation.

## Step 1: Create a POST operation in the existing path



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

The screenshot shows a user interface for creating a new API operation. The top bar indicates the method is **POST** and the path is **/Plans/estimate**. Below this, there are fields for **Add Tag**, **Summary**, **Operation ID** (with a  labeled **Deprecated**), and **Description**. The description field contains the text: **Calculate savings growth with annual compounding**. At the bottom right, there are **Edit** and **Preview** buttons.

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-16. Step 1: Create a POST operation in the existing path

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

## Step 2: Define the input parameter in the message body

Parameters						Add Parameter
Name	Located In	Description	Required	Type		
plan	Body	Savings plan	<input checked="" type="checkbox"/>	plan		

Responses			Add Response
Status Code	Description	Schema	
200	200 OK	plan-result	

- In the **POST /Plans/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 operations return the result in the same manner, the **responses** section in the operation also returns a message with a **plan-result** schema definition

Figure 3-17. Step 2: Define the input parameter in the message body

## Step 3: Define the plan schema data type

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

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-18. Step 3: Define the plan schema data type



## API definition: Source view

The screenshot shows the API Designer interface with the 'Source' tab selected. On the left, a sidebar lists API components: Extensions, Properties, Paths, /Plans/estimate, Parameters, Definitions, plan, plan-result, Services, and Tags. The main area displays the raw YAML code for the '/Plans/estimate' endpoint. The code includes definitions for paths, get methods, responses (with status 200 OK), schema (\$ref), parameters (deposit and rate), and descriptions for the deposit and interest rate.

```

52     encoded: false
53   paths:
54     /Plans/estimate:
55       get:
56         responses:
57           '200':
58             description: 200 OK
59             schema:
60               $ref: '#/definitions/plan-result'
61             parameters:
62               - name: deposit
63                 type: number
64                 required: true
65                 in: query
66                 description: Annual deposits
67                 format: float
68               - name: rate
69                 type: number
70                 required: true
71                 in: query
72                 description: Interest rate
  
```

- The OpenAPI specification supports two API definition file formats:
  - JavaScript Object Notation (JSON) text file
  - Yet Another Markup Language (YAML) text file
- IBM API Connect exports OpenAPI definitions in a YAML file format
- You can also review and edit the OpenAPI definition directly as a YAML text file in API Designer

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-19. 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 Designer, or any text editor.

## Example: savings\_1.0.0.yaml

```

swagger: '2.0'
info:
  x-ibm-name: savings
  title: savings
  version: 1.0.0
  description: Savings plan estimator to calculate growth from
    compound interest
  contact:
    name: Thomas Watson
    email: thomas@think.ibm
    url: www.ibm.com
  schemes:
    - https
basePath: /savings
consumes:
  - application/json
produces:
  - application/json
securityDefinitions: {}
security: []

```

Creating an API definition

© Copyright IBM Corporation 2017

*Figure 3-20. Example: savings\_1.0.0.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.



### Note

You can find a copy of this demonstration in the `lab_files` directory, under `demos/savings`.

## 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 2017

Figure 3-21. 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

Phase	Realized
<input checked="" type="checkbox"/> Enforced	Enforce API using a gateway
<input checked="" type="checkbox"/> Testable	Allow the API to be tested using the developer portal test tool
<input checked="" type="checkbox"/> CORS	Enable CORS access control

- 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 2017

Figure 3-22. 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 specific 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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)



## Extensions: Environment-specific properties

The screenshot shows a configuration interface for an 'app-server' extension. At the top, it says 'Property Name \* app-server' with an 'Encode' checkbox. Below that is a 'Description' field containing 'API implementation server host name'. A note says 'Define catalog specific values for this property below'. Under 'Add value', there's a table with two columns: 'Catalog' and 'Value'. A single row shows 'Default' in the Catalog column and 'https://appsvr.think.ibm/' in the Value column.

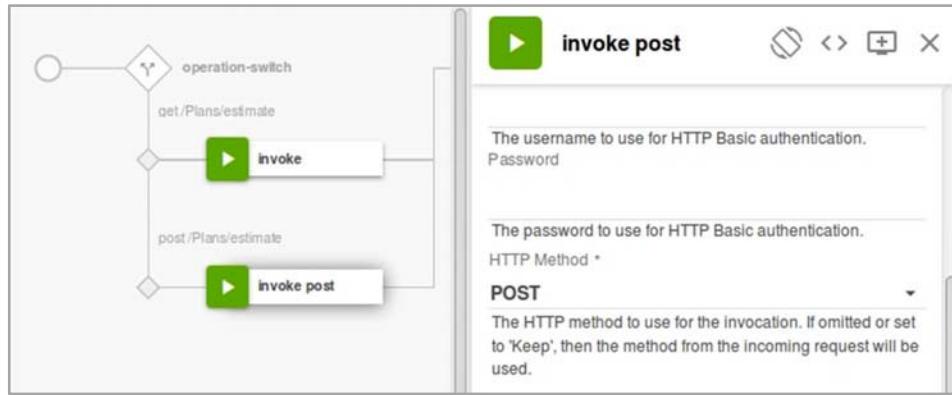
- In API Connect, catalogs separate APIs for development, testing, and production
  - You can use catalogs to group published APIs to a deployment environment
- Use **properties** extension to save catalog-specific settings in your API definition
  - For example, the **app-server** property stores the API implementation host name for the local development and sandbox testing environments

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-23. Extensions: Environment-specific properties

## 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 Designer assembly view
- You explore the concept of message processing policies in a later unit and exercise

[Creating an API definition](#)

© Copyright IBM Corporation 2017

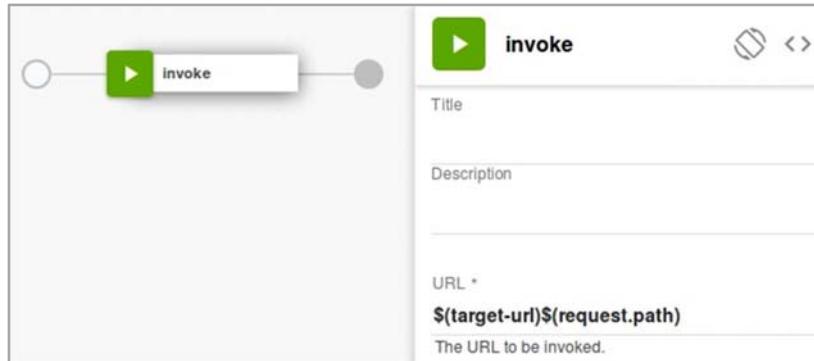
Figure 3-24. Extensions: Message processing policy assembly

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.

For example, you can log the request message header and body with the **activity log** policy.

## 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 `$(target-url)` variable represents the **host** name of the server that hosts the API implementation
  - The `$(request.path)` variable represents the web path in the original HTTP request message

Figure 3-25. Example: The invoke policy

## API templates

- You can automate the process of creating API definitions:
  - Import an existing OpenAPI document (REST) or WSDL document (SOAP)
  - Use a **template** to generate API definitions
- Template files follow the **Handlebars** format to store variables of the form `{{variable-name}}`
  - For an introduction to the Handlebars format, see: <http://handlebarsjs.com>
- To create an API definition from a template, use the `apic` command-line utility:

```
$ apic create --type api --template inventory.hbs --title "Inventory"
```

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-26. API templates

You can automate the process of creating API definitions. You can import an existing OpenAPI document (REST) or WSDL document (SOAP). Or, you can use a template to generate API definitions.

Template files follow the Handlebars format to store variables that are enclosed in two sets of curly braces.

For more information and a list of variables, see “Template variables for API and Product definitions”:

[https://www.ibm.com/support/knowledgecenter/en/SSMNED\\_5.0.0/com.ibm.apic.toolkit.doc/rapi\\_template\\_vars.html](https://www.ibm.com/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/rapi_template_vars.html)

## Example: Default API definition template

```

swagger: '2.0'
info:
  x-ibm-name: {{name}}
  title: {{title}}
  version: {{version}}

schemes:
{{#if schemes}}
  {{#each schemes}}
    - {{this}}
  {{/each}}
{{else}}
  - https
{{/if}}
host: {{hostname}}
basePath: {{basepath}}

consumes:
  - application/json
produces:
  - application/json

```

Entries in a set of double curly braces (`{{ }}`) represent an expression

- When the apic utility processes the template, it evaluates the contents of the expression

Block expressions call upon helper functions to render a section of the document

- For example, the `{{#if }}` block helper renders the section if the condition is true
- The `{{#each }}` helper iterates over a list of items in the *schemes* property

[Creating an API definition](#)

© Copyright IBM Corporation 2017

Figure 3-27. Example: Default API definition template

When you select **Add > New API** in the API Designer web application, the New API wizard generates an OpenAPI definition file based on your entries in the wizard. You enter the **name**, **title**, and **version** fields in the wizard. Optionally, you can also add a set of transport protocol schemes, the **host name**, and **base path** for the API.

The API Designer generates your API definition file from the **default API definition template**: a structured API definition file with expressions that are denoted with a double set of curly braces (`{{ }}`).

This file convention is from a project named **handlebars**. For more information about the format and processing options, see: <http://handlebarsjs.com>

In the simplest case, an expression (`{{ }}`) is a substitution variable for a runtime variable, such as **name**, **title**, and **version**. The block expression (`{{# }}`) calls helper functions that add conditional processing and looping constructs to a handlebars template file. In this example, the template iterates over a list of transport protocol schemes. If the user did not specify any schemes, add `https` as a default scheme for the API definition.

To review the entire default API definition template, see:

[http://www.ibm.com/support/knowledgecenter/en/SSMNED\\_5.0.0/com.ibm.apic.toolkit.doc/rapim\\_template\\_examples.html](http://www.ibm.com/support/knowledgecenter/en/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/rapim_template_examples.html)

## Unit summary

- Explain the concept of an API definition
- Explain the purpose of the OpenAPI specification
- Document the API version and description
- Define an API operation
- Define query and path parameters
- Define request and response headers
- Explain the purpose of API properties
- Define environment variables as properties
- Modify the target URL as a property
- Explain the concept of API templates

[Creating an API definition](#)

© Copyright IBM Corporation 2017

*Figure 3-28. Unit summary*

## 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: The **path** parameter consists of an HTTP method and a resource name.



Creating an API definition

© Copyright IBM Corporation 2017

Figure 3-29. 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: The **path** parameter consists of an HTTP method and a resource name.

The answer is False. The **path** parameter is the web resource that follows immediately after the base path. It does not include an HTTP method.



## Exercise: Creating and publishing an API in API Designer

Creating an API definition

© Copyright IBM Corporation 2017

Figure 3-31. Exercise: Creating and publishing an API in API Designer

## Exercise objectives

- Create an API definition in the API Designer
- Review the operations, properties, and data types in an API definition
- Publish an API to the API Connect development environment
- Start an API and the micro gateway
- Test an API with the API Explorer



[Creating an API definition](#)

© Copyright IBM Corporation 2017

*Figure 3-32. Exercise objectives*

---

# Unit 4. Defining APIs that call REST and SOAP services

## Estimated time

01:00

## Overview

With API Designer, 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 SOAP service in an API definition
- Explain how to expose an existing REST service in an API definition

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

*Figure 4-1. Unit objectives*

## 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, found in service-oriented architecture (SOA) and enterprise service bus (ESB) infrastructure
  - You map functions to SOAP operations
  - Applications call API operations in an XML-based SOAP message format
  - Applications exchange data in XML
- The third category, **OAuth 2.0 Provider**, defines a REST API that acts as an authorization and token server in the OAuth 2.0 message flow
  - You examine the OAuth 2.0 specification in detail later in this course

[Defining APIs that call REST and SOAP services](#)

© Copyright IBM Corporation 2017

Figure 4-2. What types of APIs can you define?

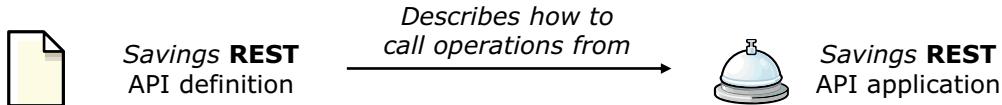
You can define two types of APIs in the API Designer.

- 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.

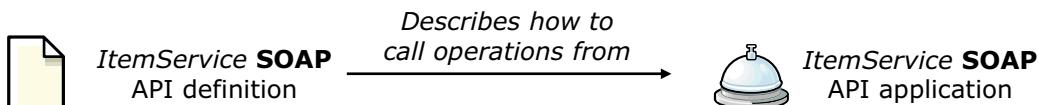
The API Designer can also create OAuth 2.0 Providers: A REST API that is used in the OAuth 2.0 authorization message flow. You examine the OAuth 2.0 specification in detail later in this course.

## Scenarios

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



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



- In later units, you learn how to define a REST API that calls a SOAP API with message processing policies

Figure 4-3. 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 specify its operations, request, and response messages in an OpenAPI definition with the API Designer application.

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 2017

*Figure 4-4. Proxy an existing REST API*

## Topics

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

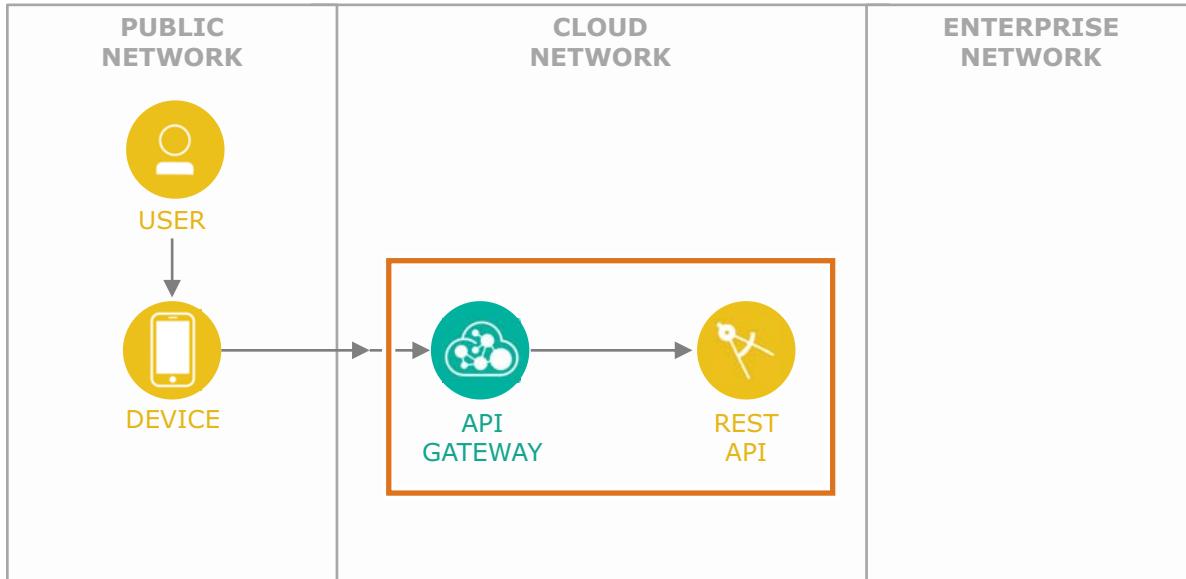
Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

*Figure 4-5. Topics*

## Overview: Proxy an existing REST API

- In this scenario, you define an API that calls an existing REST API



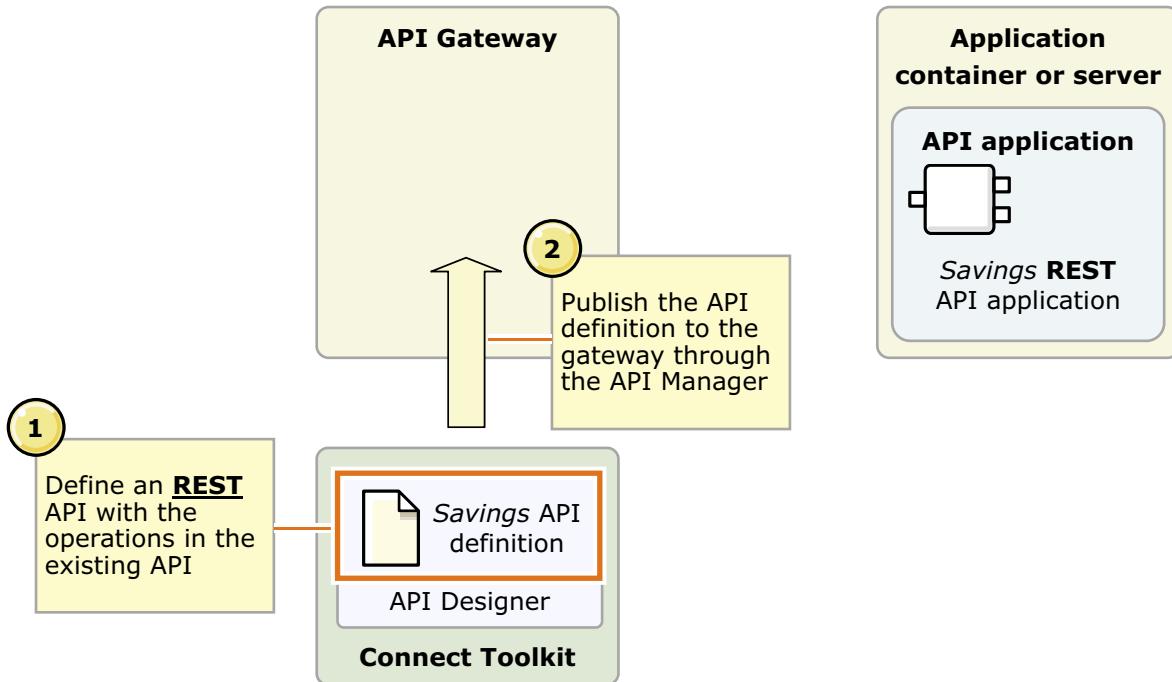
Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

*Figure 4-6. Overview: Proxy an existing REST API*

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. The following slides explain how to define an API definition for an existing REST endpoint. When you publish the API to your API Connect Cloud, the API gateway manages access to the API application.

## Scenario one: Define a REST API



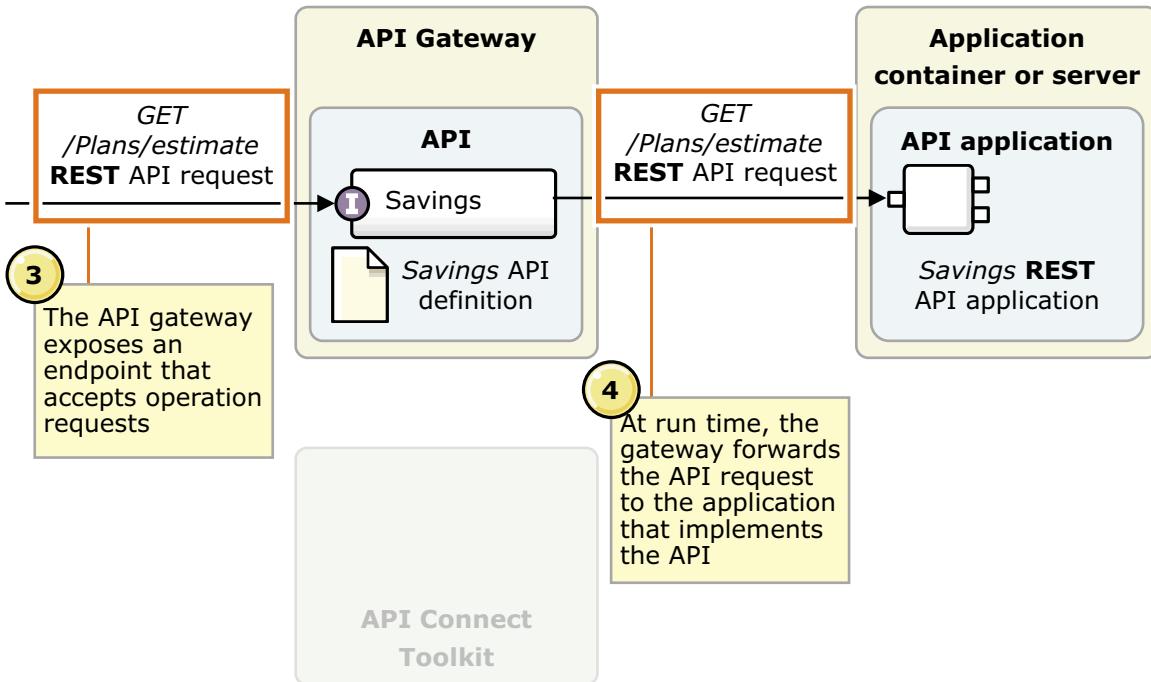
Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-7. Scenario one: Define a REST API

In the first scenario, you want to define a REST API that calls an existing REST application. In the last lab exercise, you defined an OpenAPI document that describes the operation from the Savings REST API application. You defined a REST API in the API editor, and you saved the interface as an OpenAPI definition. You published the API definition to the API Manager, which makes the service available on the API gateway.

## Scenario one: Gateway forwards REST API requests



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-8. 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 as-is to the endpoint address with the application that 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 /Plans/estimate` API path and operation call to the implementation at the `savingsample.mybluemix.net` website.

## Step 1: Create a REST API definition in API Designer

1. Create a directory for the project
2. Start the **API Designer** web application
3. Create an OpenAPI definition from scratch
  - If the existing API implementation already has an OpenAPI definition, you can import it directly into API Designer

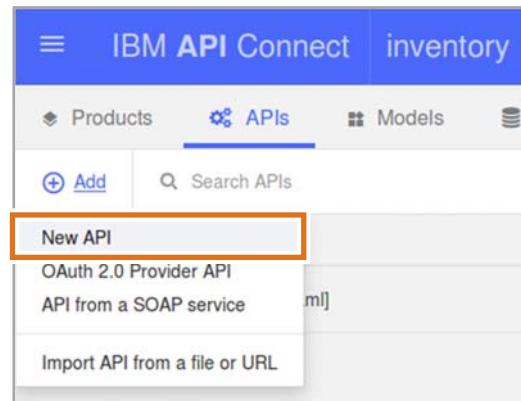


Figure 4-9. Step 1: Create a REST API definition in API Designer

To create an OpenAPI definition, create a directory in your workstation to hold the definition files. Start the API Designer web application from the directory. In the API Designer application, select **New OpenAPI from scratch** to create an empty definition file.

If the REST API implementation already has an OpenAPI definition, you can select **Import an existing OpenAPI** to use the API definition as a starting point.

## Step 2: Define the base path and target endpoint

New API

Title \*  
savings

Name \*  
savings

Base Path \*  
**/api**

Version \*  
1.0.0

**API template**

Create API using template  
Default

**Target**

Target endpoint (if known)  
**http://savingsample.mybluemix.net**

**Security**

Identify using  
None

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

1. Enter a name, description, contact information, 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. In the additional properties section, specify the **target endpoint** for the API as the network location of the existing API implementation

Figure 4-10. 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 additional properties 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 area of additional properties, the value is placed in the host field of the YAML file that gets generated for the API.

## Step 3: Define the API operations

savings 1.0.0	GET /Plans/estimate			
Operations				
GET /Plans/estimate	Description			
POST /Plans/estimate	Calculate savings growth with annual compounding			
Definitions	Parameters			
plan	Name	Located in	Required	Schema
plan-result	deposit	query	true	number
	Annual deposits			
	rate	query	true	number
	Interest rate			
	years	query	true	integer
	Years of saving			
	Responses			
	Code	Schema		
	200	plan-result		

If you created an OpenAPI definition from scratch, you must define the paths, HTTP method, request, and response messages for each API operation

1. Define one or more **paths** for each resource in the API
2. Define the operations, request, and response messages to match the existing REST API

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-11. Step 3: Define the API operations

You documented the metadata and base path for the API definition. You also specified the location of the API implementation. To complete the API definition, you must specify the exact interface for the API. The interface includes the API paths, HTTP method, request, and response message for each API operation.

For example, the **GET /Plans/estimate** operation has a path of **/Plans/estimate**. This path is mapped to the HTTP **GET** method. The operation takes three input parameters in the request: **deposit**, **rate**, and **years** as query parameters. The types of the three input parameters are number-float for **deposit**, number-float for **rate**, and integer-int64 for **years**. The expected response message is an HTTP status code of 200, with a value of **balance** in the message body.

If you want to hide an API operation that is available in the API implementation, omit the API operation from the OpenAPI definition.

## Step 4: 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



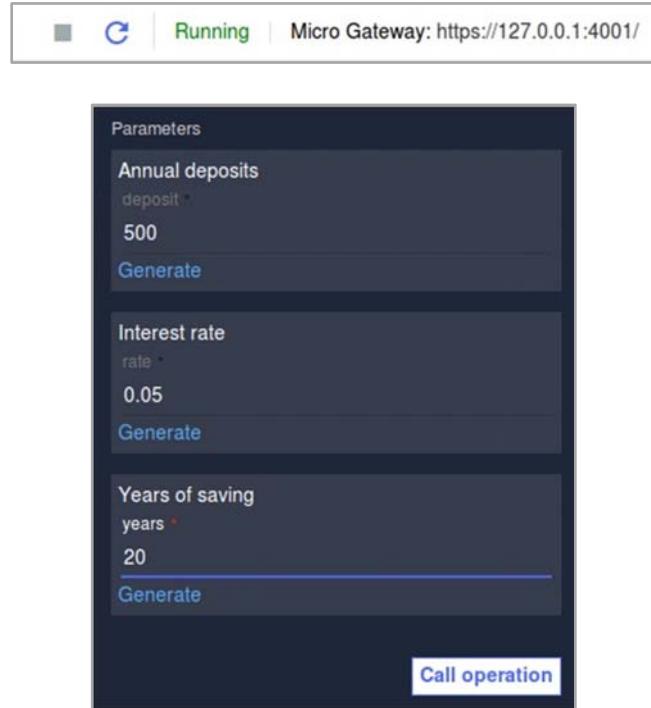
Figure 4-12. Step 4: Configure the gateway to call the REST application

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 Designer 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 Designer 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 specified in API Designer. The **request path** context variable is the API path name for the operation.

## Step 5: Test the API definition

- To test the REST API definition, start the micro gateway
  - The API designer publishes the API definition when it starts the gateway
  - You do not need to explicitly publish an API application for local testing in the micro gateway
- Review the API operations in the **Explore** view
  - Copy the sample code to call the published API endpoint
  - Use the test client to send a request to an API operation



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-13. Step 5: Test the API definition

To test the API endpoint that you defined in the OpenAPI definition, you must publish the API to a gateway. In this case, both the micro gateway and DataPower gateway support REST APIs with a single invoke policy in their assembly.

To test the REST API in the local micro gateway, start the gateway from the application bar at the bottom of the API Designer web application. By default, the micro gateway listens to API requests at port 4001.

You can review the documentation and sample code for the API definition in the **Explore** view. Select an API operation, and enter or generate sample input parameter values in the test client. Select **Call operation** to call 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 2017

*Figure 4-14. Proxy an existing SOAP API*

## Topics

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

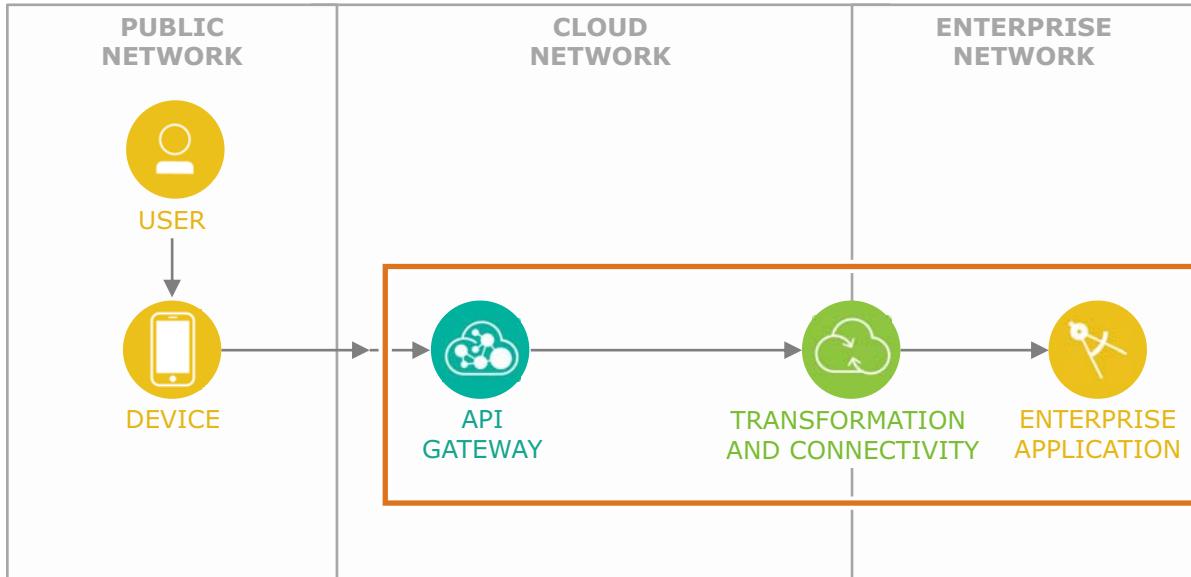
Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

*Figure 4-15. Topics*

## Overview: Proxy an existing SOAP API

- In this scenario, you define an API that calls an existing SOAP API



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

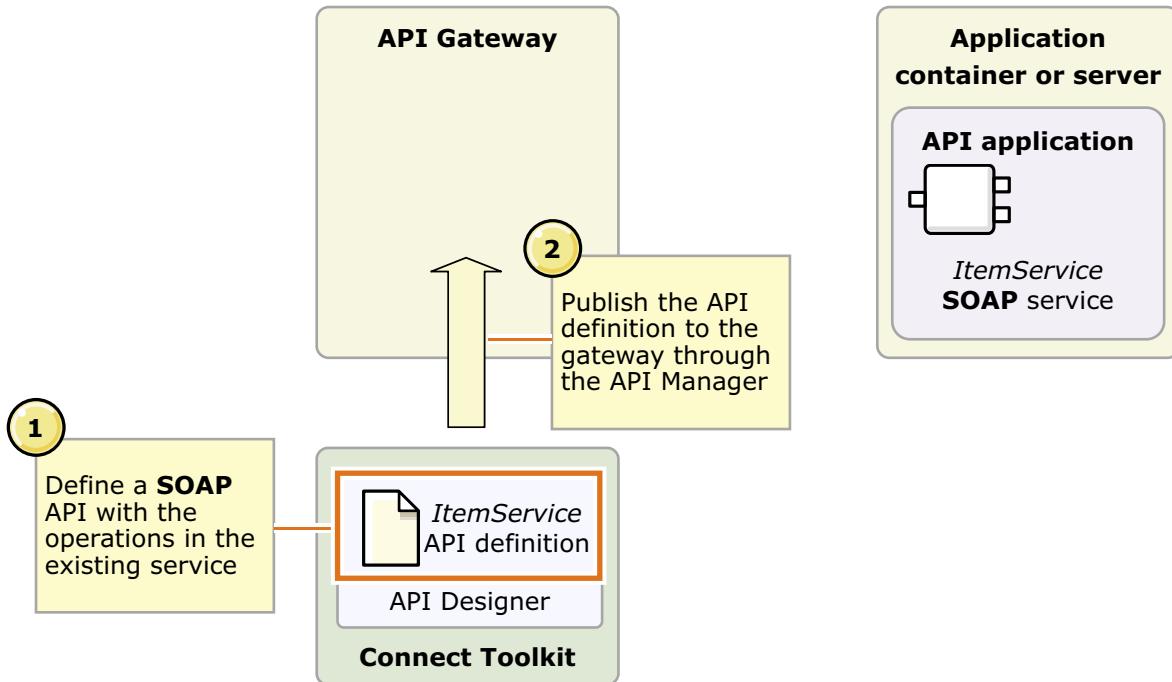
Figure 4-16. Overview: Proxy an existing SOAP API

SOAP services are commonly found in enterprise application networks. Although more recent API implementations follow the REST architecture, many organizations still use and support SOAP style services.

The API gateway in IBM API Connect can directly bind and call SOAP services: it can act as a client to SOAP APIs. When you define an API that calls a SOAP endpoint, the API Designer adds a “proxy” object to call a SOAP service.

The API gateway can directly call a SOAP service as part of an enterprise application. It can also route a SOAP service call through a service bus, which transforms and connects to the actual service implementation. For simplicity’s sake, the following slides focus on the connection between the API gateway and the SOAP enterprise application.

## Scenario two: Define a SOAP API



Defining APIs that call REST and SOAP services

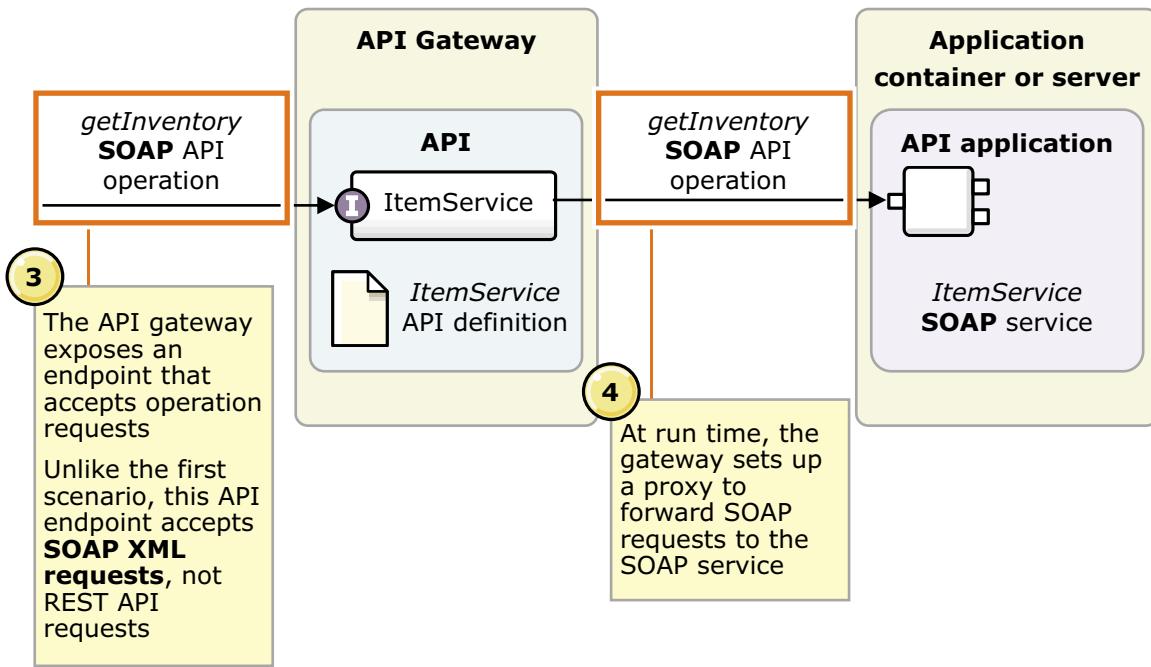
© Copyright IBM Corporation 2017

Figure 4-17. 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 Designer web application, build a new OpenAPI definition that is based on a SOAP interface. The most straightforward method is 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 publish the API definition to a DataPower gateway through the API Manager.

## Scenario two: Gateway forwards SOAP API requests



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-18. 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 Designer Explore 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. Only the DataPower gateway supports the SOAP web service proxy: you cannot deploy a SOAP API definition to the micro gateway.

## 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 2017

Figure 4-19. 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.

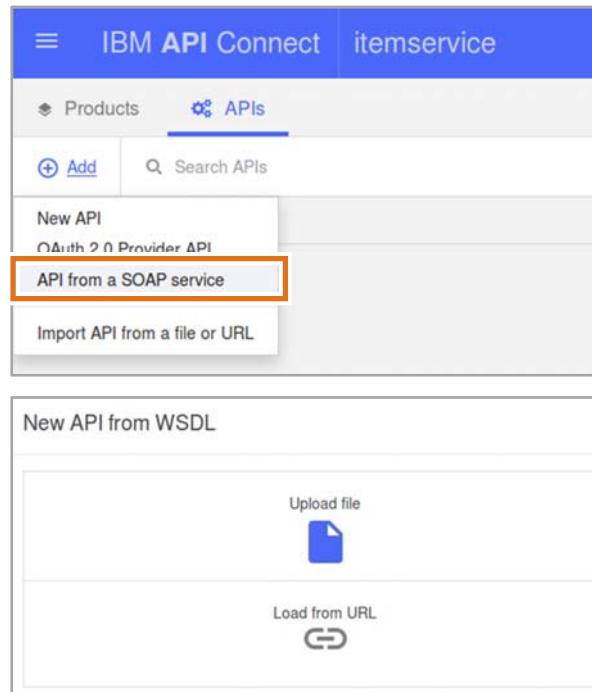
Notice that SOAP services have a specific way to format input parameters and the operation response. The SOAP specification defines an XML message format that is known as the **SOAP** envelope that encapsulates parameters, results, and fault messages. The OpenAPI definition must define how to create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

1. Create a directory for the project
2. Start the **API Designer** web application
3. Create an Open API from a SOAP service
4. Import a copy of the WSDL document from your workstation or on a server

- The suggested practice is to create an SOAP API definition from an existing WSDL document



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-20. Step 2: Create a SOAP API definition in API Designer

Before you begin, create a directory to hold the API definition file for the SOAP API. Start the API Designer web application from the directory that you created. Select the **New OpenAPI from SOAP** service option to create an OpenAPI document based on an existing WSDL document.

You have two options: either upload the WSDL document from a local directory, or retrieve the WSDL document from a web server. In this example, you download a copy of the ItemService WSDL document from the <http://soapsample.mybluemix.net/ItemService?WSDL> link. Upload the WSDL document from the copy that you saved in your workstation.

In theory, you can define an OpenAPI definition for a SOAP API from scratch. However, you must define each WSDL port type, operation, input, output, and fault message exactly as described in the WSDL document. The suggested practice is to import the WSDL document that describes the SOAP service into API Designer. The web application generates the proper OpenAPI paths, operations, and schema types based on the WSDL document.

## Step 3: Define the metadata and target endpoint

- After the New API from WSDL wizard parses the document, select the SOAP service that you want to define as an API
  - You can also select specific operations to expose in the API
- In the same manner as a REST API definition, provide the name, description, contact information, and version of the API

The figure consists of two vertically stacked screenshots of the 'New API from WSDL' wizard in the IBM API Designer.

**Screenshot 1: Selecting the SOAP Service**

- The title is 'New API from WSDL'.
- The WSDL file is 'ItemService.wsdl'.
- 1 SOAP service found: 'ItemService'.
- Operations: 'getInventory'.
- Buttons: 'Back', 'Cancel', 'Add a product...', 'Done'.

**Screenshot 2: Defining API Product Metadata**

- The title is 'New API from WSDL'.
- Product template: 'Create product using template' (Default).
- Info:
  - Title: 'soap-services'
  - Name: 'soap-services'
  - Version: '1.0.0'
- Publishing: 'Publish this product to a catalog' checkbox.
- Buttons: 'Back', 'Cancel', 'Done'.

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-21. Step 3: Define the metadata and target endpoint

The **New API from WSDL** wizard in the API Designer web application parses the contents of the WSDL document. You have the choice to select one or more SOAP operations from the WSDL document. API Designer creates an API path and operation for each SOAP operation that you select.

In this example, the **ItemService** SOAP service has one operation: **getInventory**. Select the ItemService SOAP service check box to import the getInventory operation into the OpenAPI definition.

After you convert a SOAP service, you complete the OpenAPI definition metadata: the name, description, contact information, and version number for the API.

You can add the API definition into an API product. You explore the concept of API products in a later unit in this course.

## Step 4: Review the host, base path, and media type

Host	Host *	
Base Path	Base Path	/ItemService
Consumes	<input type="checkbox"/> application/json <input type="checkbox"/> application/xml Additional media types text/xml <input checked="" type="checkbox"/> Add media type	
Produces	<input type="checkbox"/> application/json <input checked="" type="checkbox"/> application/xml	

- The WSDL import wizard defines the following settings:
  - The **target-url** setting is empty
  - The **base path** is set to the name of the SOAP service operation
- SOAP APIs send request and response messages in XML format, in contrast to JSON data types in REST APIs
  - The **consumes** and **produces** properties are set to the **text/xml** and **application/xml** media types

Figure 4-22. Step 4: Review the host, base path, and media type

The **Import OpenAPI from SOAP** wizard defines a **base path** based on the name of the SOAP service operation. In effect, the base path at the API gateway and the actual API implementation have the same value. The **target endpoint** value is empty. However, the API definition does not use this value to create a proxy from the API gateway to the API implementation.

Unlike REST APIs, SOAP APIs use XML data types in the HTTP request and response messages. The wizard sets the **consumes** and **produces** sections of the API definition to the **text/xml** and **application/xml** media types. In general, the **text/xml** media type is less restrictive than **application/xml**. Both media types indicate that the message body uses XML encoding. The **application/xml** media type specifies that an application created the XML data, rather than a form on a web page.

## Step 5: Review the API operations

- The API Designer 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 'getInventory' API operation configuration in the IBM API Designer. The top bar indicates the method is POST and the path is /getInventory. Below this, there are sections for 'Add Tag', 'Summary' (Operation getInventory), 'Operation ID' (getInventory), and a 'Deprecated' checkbox which is unchecked. A 'Description' field is present with an 'Edit' and 'Preview' link. The 'Parameters' section has a header row with columns: Name, Located In, Description, Required, and Type. A single parameter 'body' is listed, located in 'Body', required, and of type 'getInventoryInput'. There are also 'Add Parameter' and delete icons.

Figure 4-23. Step 5: Review the API operations

Review the API paths and operations. For each operation in the WSDL port type, the wizard 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 6: Review the request and response type definitions

Definitions

Security

```

getInventoryInput:
type: object
properties:
  Envelope:
    xml:
      prefix: soap-env
      namespace: 'http://schemas.xmlsoap.org/soap/envelope/'
    type: object
    properties:
      Header:
        $ref: '#/definitions/getInventoryHeader'
      Body:
        type: object
        properties:
          getInventory:
            $ref: '#/definitions/getInventory'
example: |-
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
<soap-env:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<UsernameToken>
<Username>string</Username>
<Password>string</Password>
</UsernameToken>
</wsse:Security>
</soap-env:Header>
<soap-env:Body>
<tns:getInventory xmlns:tns="http://soapsample.training.ibm.com/"></tns
: getInventory>
</soap-env:Body>
</soap-env:Envelope>

```

getInventoryOutput

getInventoryHeader

getInventory

getInventoryResponse

item

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-24. Step 6: 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. Review the inline examples in the OpenAPI definition to see the sample API operation request.

## Example: The item XML schema type

```

    <xs:complexType name="getInventoryResponse">
        <xs:sequence>
            <xs:element form="unqualified" maxOccurs="unbounded" minOccurs="0" name="return" type="tns:item"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="item">
        <xs:sequence>
            <xs:element minOccurs="0" name="name" type="xs:string"/>
            <xs:element minOccurs="0" name="description" type="xs:string"/>
            <xs:element minOccurs="0" name="img" type="xs:string"/>
            <xs:element minOccurs="0" name="img_alt" type="xs:string"/>
            <xs:element minOccurs="0" name="id" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

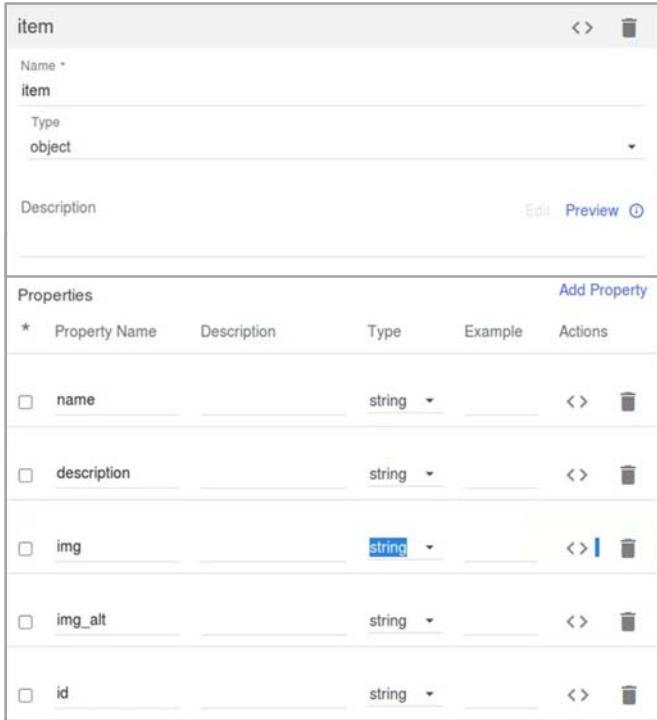
```

- This example shows a portion of the WSDL document that you imported into the OpenAPI definition
- The `getInventory` operation returns a list of items
  - The “item” complex type describes the properties in an item record

*Figure 4-25. Example: The item XML schema type*

The start of this slide displays the original WSDL schema type for the `getInventoryResponse` message. The response message returns a list of inventory items as a complex XML type with five properties: name, description, image, alternative image, and identifier.

## Example: The item API schema definition



The screenshot shows the 'item' schema definition in the IBM API Designer. The schema has a name 'item', type 'object', and properties: name, description, img, img\_alt, and id.

Property Name	Description	Type	Example	Actions
<input type="checkbox"/> name		string		<input type="button" value="Add Property"/> <input type="button" value="Edit"/> <input type="button" value="Preview"/> <input type="button" value="Delete"/>
<input type="checkbox"/> description		string		<input type="button" value="Add Property"/> <input type="button" value="Edit"/> <input type="button" value="Preview"/> <input type="button" value="Delete"/>
<input type="checkbox"/> img		string		<input type="button" value="Add Property"/> <input type="button" value="Edit"/> <input type="button" value="Preview"/> <input type="button" value="Delete"/>
<input type="checkbox"/> img_alt		string		<input type="button" value="Add Property"/> <input type="button" value="Edit"/> <input type="button" value="Preview"/> <input type="button" value="Delete"/>
<input type="checkbox"/> id		string		<input type="button" value="Add Property"/> <input type="button" value="Edit"/> <input type="button" value="Preview"/> <input type="button" value="Delete"/>

- The OpenAPI definition captures the “item” complex type as a schema definition

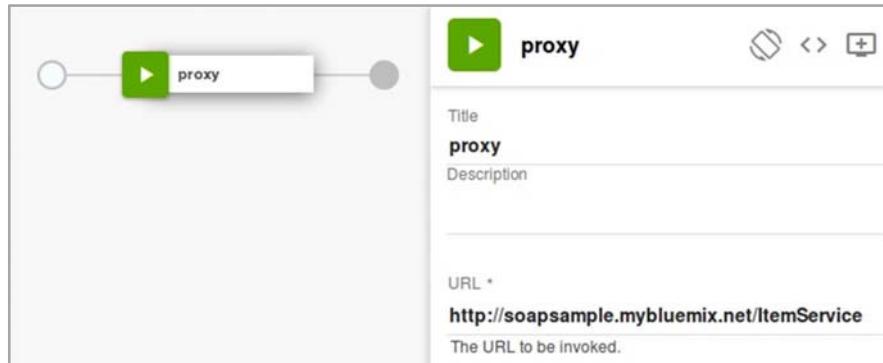
Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-26. Example: The item API schema definition

This slide shows the same **item** XML type that is stored as an OpenAPI schema definition.

## Step 7: Review the proxy policy in the assembly flow



- The **proxy** policy forwards the original SOAP API request to the actual SOAP service implementation
- The API Designer sets the **target URL** setting in the **proxy** policy to the WSDL service endpoint address
  - For example, <http://soapsample.mybluemix.net/ItemService> is the address for the ItemService SOAP API implementation
  - You can modify this setting after you import the WSDL document

*Figure 4-27. Step 7: Review the proxy policy in the assembly flow*

The **proxy** policy is designed to forward the message payload in the assembly flow to the **URL** endpoint location. This policy is similar to the **invoke** policy, but it has two distinct differences. First, the API gateway can run one **proxy** policy per flow – you cannot call multiple **proxy** policies in the same flow. Second, this policy is unique to the DataPower gateway. You cannot publish an OpenAPI definition with a proxy policy to a micro gateway.

## Step 8: Start the toolkit Docker DataPower gateway

- Unlike REST APIs, you must test SOAP APIs with a DataPower gateway
  - The **proxy** policy is not available in the micro gateway
  - The OpenAPI definition for the API contains the statement:  
gateway: datapower-gateway
- To use the DataPower gateway, you must be running at least v5.0.7 of the API Connect toolkit and Docker must be installed
- Start the toolkit DataPower gateway from the directory that stores the API with the command:  
`apic start`

```
$ apic start
Service itemservice-gw starting,
use "apic services" to obtain
port details.
$ apic services
Service itemservice-gw running
on port 4001.
```

```
$ docker ps
CONTAINER ID        IMAGE
5cb190c959e0        ibm-
apiconnect-toolkit/datapower-
api-gateway:1.1.8

f943df03efc7        ibm-
apiconnect-toolkit/datapower-
mgmt-server-lite:1.1.8
```

Figure 4-28. Step 8: Start the toolkit Docker DataPower gateway

Only the DataPower gateway supports the SOAP web service proxy: you cannot deploy a SOAP API definition to the micro gateway. The micro gateway in your local workstation does not support the **proxy** policy action.

When you test API from the Developer toolkit, you can now set an option to use the DataPower gateway Docker container for a full set of security and policy capabilities. The toolkit synchronizes with the gateway when you save an API that uses DataPower gateway policies. The OpenAPI definition for the API contains the statement **gateway: datapower-gateway**.

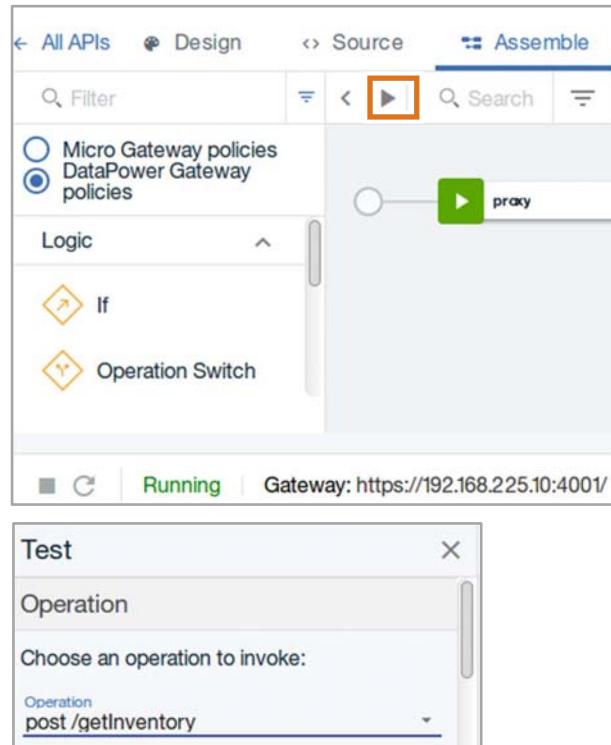
Use the command `apic start` to download the Docker images and start the DataPower gateway as a Docker container.

If you are running the API Connect toolkit at version 5.0.7 or later, and Docker is installed on the same workstation, then you can test your SOAP API on the embedded DataPower gateway.

Start the toolkit DataPower gateway from the directory that contains the API with the command:  
`apic start`

## Step 9: Test the SOAP API in the Designer toolkit

- When the DataPower gateway is started, test the SOAP API from the test feature
- 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



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-29. Step 9: Test the SOAP API in the Designer toolkit

When the gateway displays as running in the API Designer, test the SOAP API by clicking the Test icon in the assembly editor.

Then, 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 10: Response message from the SOAP API call

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

**Test**

Response

Status code:  
200 OK

Response time:  
1834ms

Headers:

```
content-language: en-US
content-type: text/xml; charset=ISO-8859-1
x-global-
transaction-id: f21b30f25a00ec7400004132
```

Body:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns2:getInventoryResponse
            xmlns:ns2="http://soapsample.training.ibm.com">
            <return>
                <ns2:name>Dayton Meat Chop</ns2:name>
            </return>
        </ns2:getInventoryResponse>
    </soap:Body>
</soap:Envelope>
```

Figure 4-30. Step 10: 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 SOAP service in an API definition
- Explain how to expose an existing REST service in an API definition

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

*Figure 4-31. Unit summary*

## Review questions

1. True or False: The micro gateway supports SOAP and REST API definitions.
2. How do you define an SOAP API Definition in API Designer?
  - A. Import a WSDL document in API Designer
  - 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 2017

Figure 4-32. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: The micro gateway supports SOAP and REST API definitions.  
The answer is False. The micro gateway does not support SOAP API definitions.
2. How do you define an SOAP API Definition in API Designer?
  - A. [Import a WSDL document in API Designer](#)
  - B. Create a SOAP API from the assembly palette
  - C. Generate a sample with the apic utility
  - D. Import a SOAP application in API DesignerThe 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 2017

Figure 4-33. Review answers



## Exercise: Defining an API that calls an existing SOAP service

Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-34. Exercise: Defining an API that calls an existing SOAP service

## Exercise objectives

- Create an API definition with the API Designer web application
- Define API operations that map to a SOAP web service
- Map SOAP message types to API data types
- Test the SOAP API on the DataPower gateway of the developer toolkit
- Publish the API to API Manager
- Test the SOAP API in the API Manager test client (optional)



Defining APIs that call REST and SOAP services

© Copyright IBM Corporation 2017

Figure 4-35. 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 Node.js. You examine the structure and components in a LoopBack application, and learn how to generate an application scaffold with the Developer Toolkit.

## How you will check your progress

- Review questions
- Lab exercise

## Unit objectives

- Explain the concept of model-driven design
- Explain the purpose of the LoopBack Node.js framework
- Identify the components of a LoopBack application
- Create the application scaffold from the apic command-line utility
- Create a LoopBack application in the API Designer web application

## 5.1. Introduction to the LoopBack framework

## Introduction to the LoopBack framework

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

*Figure 5-2. Introduction to the LoopBack framework*

## Topics

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

## 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
- 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

Figure 5-4. 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 application 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. Examples of service policies include access control, quality of service assurances, message-level security, and activity logging.

## 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**
- **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

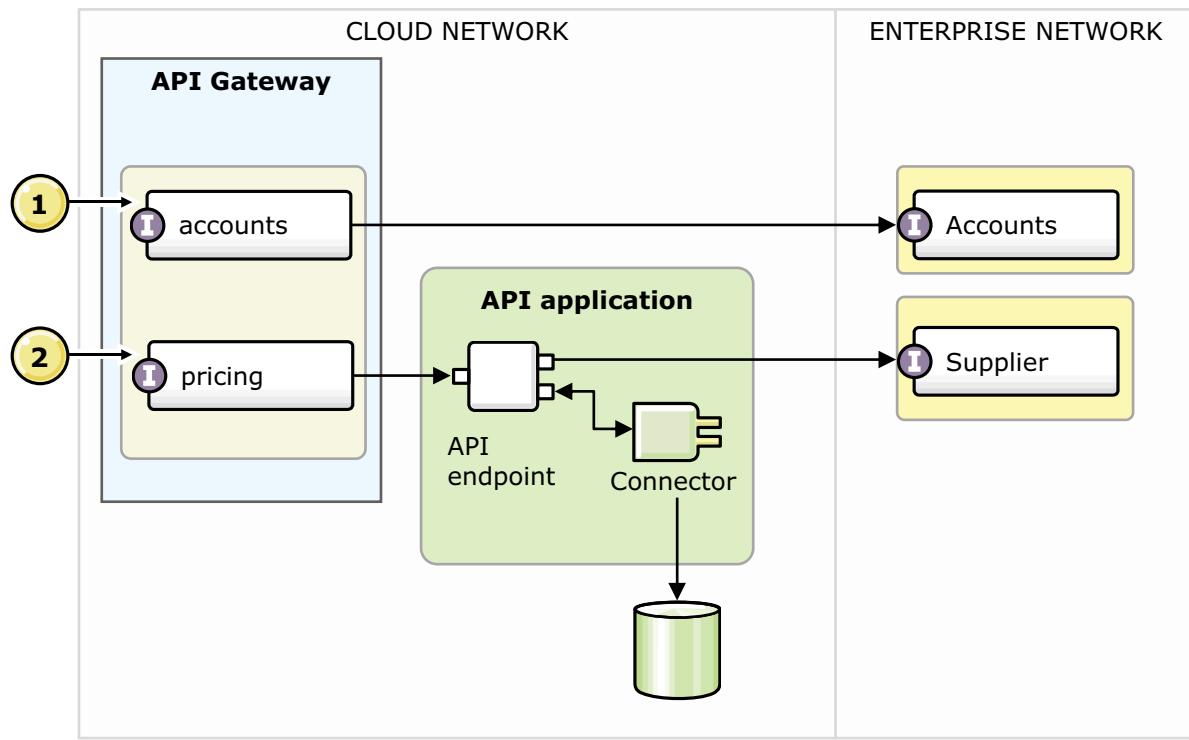
Figure 5-5. Example: Product inventory and price calculator

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 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 2017

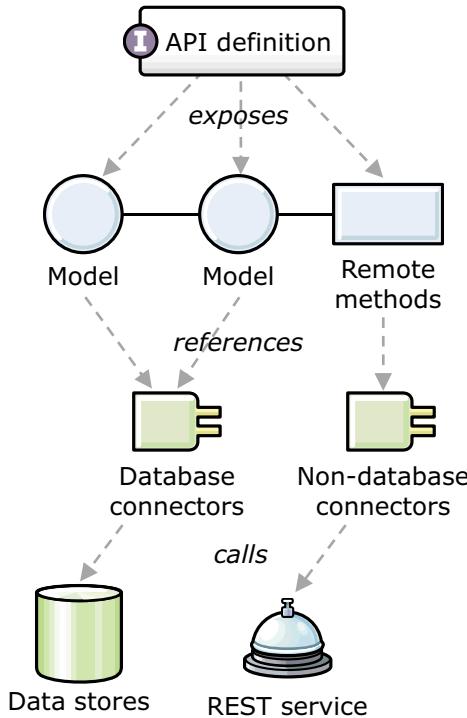
Figure 5-6. Message flows in IBM API Connect

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 2017

Figure 5-7. 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.

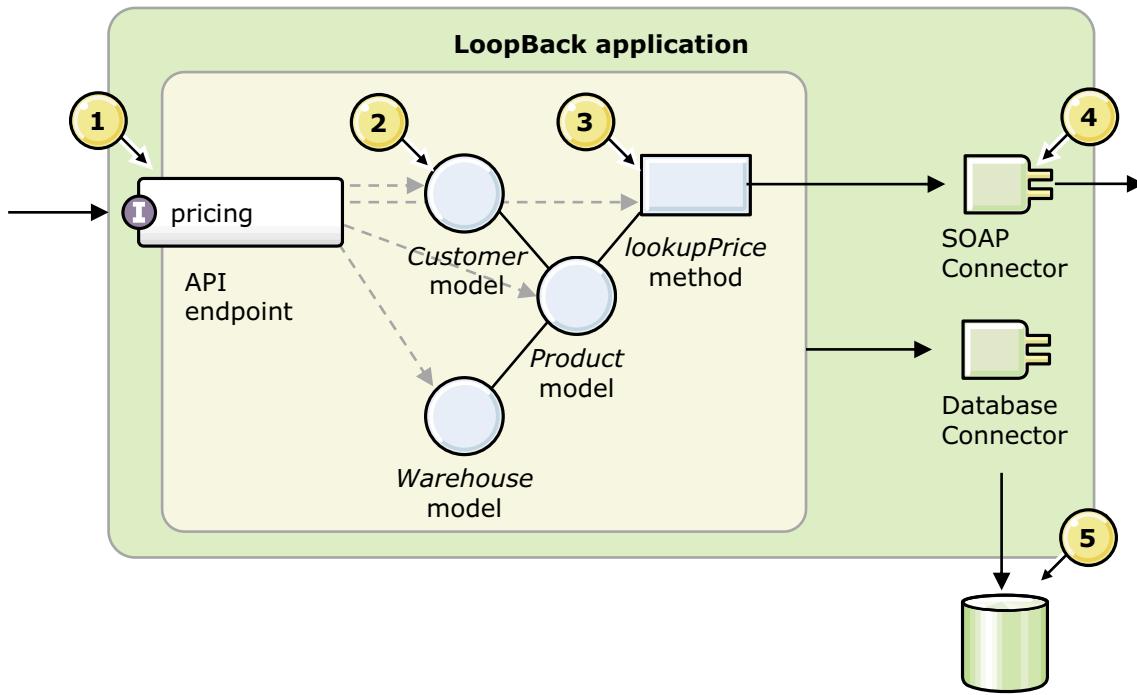
With LoopBack, you define business models, properties, and relationships through command-line tools. The LoopBack framework creates a set of predefined operations to create, retrieve, update, and delete models. The framework also persists the model data to a data source through a database connector.

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.

## Implement API with a Node.js LoopBack application



Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

Figure 5-8. Implement API with a Node.js 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 model in your application. The dashed lines denote the link between API operations and the model objects.
3. Not all API operations fall in the category of create, retrieve, update, and delete. To create your own operation, define a **remote method** within a model object.
4. Your LoopBack application does not work in isolation: it 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. The LoopBack framework can automatically persist and retrieve data for your model objects through a data source. You configure a **database connector** and register your models to the data source.

## 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 2017

*Figure 5-9. Implement an API with a LoopBack application*

## Topics

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

## Instructor demonstration



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

1. Generate a LoopBack application with the API Connect toolkit
2. Define data sources that supply information to your application
3. Create the models, properties, and relationships in the LoopBack application
4. Implement the API logic in the model object
5. Update the API definition
6. Start the API Designer web application
7. Review the API definition in the API Designer
8. Review and test the data source in the API Designer
9. Start the LoopBack application and micro gateway
10. Test the API with the API Explorer

Figure 5-11. Instructor demonstration

In this topic, the instructor demonstrates how to implement an API with the IBM API Connect Toolkit.



### Note

You can find a copy of this demonstration in the `lab_files` directory, under `demos/pricing`.

## 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, Mac OS) or command prompt (Windows)
2. Run the LoopBack application generator

```
$ apic loopback
? 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
```

[Implementing APIs with the LoopBack framework](#)

© Copyright IBM Corporation 2017

Figure 5-12. 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, Mac OS) or command prompt (Windows).
2. Run the LoopBack application generator.
3. Change directory into your LoopBack application.

The `apic loopback` 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 2a: 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

```
$ apic create --type 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@^2.0 Yes
```

2. Review and edit the data source settings

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

*Figure 5-13. Step 2a: Define a data source*

### Step 2a: 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 2b: Configure the data source

- In this example, you map the existing REST service to the LoopBack REST data source in `~/rest-connector/server/datasources.json`

```
{
  "priceREST": {
    "name": "priceREST",
    "baseURL": "https://pricesample.mybluemix.net/price/",
    "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 2017

Figure 5-14. Step 2b: Configure the data source

### Step 2b: Configure the 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>



## Syntax

```
{  
  "priceREST": {  
    "name": "priceREST",  
    "baseURL": "https://pricesample.mybluemix.net/price/",  
    "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 3: 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 create --type 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 5-15. Step 3: Create models, properties, and relationships

### Step 3: 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 create` command to generate a LoopBack model in the terminal.

## Step 4: Implement API logic

- Implement the message processing logic in the API in Node.js
  - Overwrite the code in  
~/rest-connector/common/models/new-model.js

```
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**

Figure 5-16. Step 4: Implement API logic

### Step 4: 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.



## 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: 'id', type: 'number' }],
    returns: { arg: 'price', root: true },
    http: { verb: 'get' }
  });
};
```

## Step 5: Update the API definition file

- IBM API Connect defines API interfaces in an API definition
  - When you create LoopBack components with the **apic** command-line utility, it updates the API definition file
  - You can also review and edit the API definition file with the **API Designer** web application

1. Refresh the API definition file with the changes that you made to the LoopBack model

```
$ apic loopback:refresh
Updating swagger and product definitions

Created /home/student/pricing/definitions/pricing.yaml swagger
description
```

*Figure 5-17. Step 5: Update the API definition file*

### Step 5: Update the API definition file.

The **API definition** represents the interface for your API. The paths, headers, data types, request, and response messages of each API operation are specified in the API definition. In addition, the API definition contains implementation details, such as security requirements, messaging processing policies, and environment-specific variables.

IBM API Connect saves the API definition according to the Swagger V2.0 specification YAML document.

The `loopback:refresh` command is crucial to keeping your API interface definition consistent with your API implementation. When you create a model, a property, or a relationship with the `apic` command-line utility, it updates the Swagger API definition for you. However, if you manually edit the model configuration files, or if you write a custom remote method, these changes do not automatically appear in the Swagger file. You must call `apic loopback:refresh` to update the Swagger file with your changes.

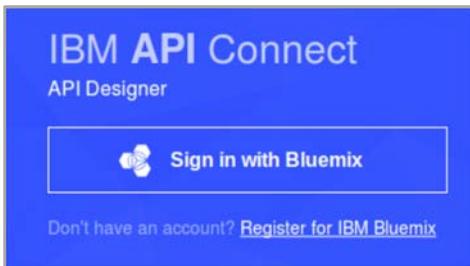
## Step 6: Start the API Designer application

- The API designer is a web application for API developers:
  - Edit the API interface in the Design view
  - Review and edit LoopBack models, properties, data sources
  - Start the LoopBack application and test Micro Gateway
  - Run an API test client against the micro gateway and LoopBack application

- Start the API Designer application

```
$ apic edit
```

- Sign in to the Designer with your Bluemix account



Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

*Figure 5-18. Step 6: Start the API Designer application*

Step 6: Start the API Designer application.

The API designer is a web application for API developers.

In the first step, open the terminal on Linux and Mac OS operating systems, or the command prompt on Windows operating systems. Enter the command `apic edit` in the prompt.

Although you installed the API Connect Toolkit locally on your workstation, the API Designer web application prompts you to log in with your IBM Bluemix account on first use. The API Designer web application uses your Bluemix credentials to deploy API Connect resources to a Cloud-enabled version of API Connect.



## Step 7: Review the API definition file with API Designer

Review the API definition in the API Designer

The screenshot shows the IBM API Connect API Designer interface. The top navigation bar includes 'IBM API Connect', 'pri ... 1.', 'Publish', 'Explore', and a help icon. Below the navigation is a toolbar with icons for 'Checklist' (highlighted), 'File', and 'More'. The main area has tabs for 'All APIs' (highlighted) and 'Design' (boxed and circled with number 1). The 'Design' tab has sub-sections: 'Paths', 'Analytics', 'Parameters', 'Definitions', 'x-any', and 'Services' (circled with number 2). The 'Paths' section displays three API paths: '/products/price/{id}', '/products/invoke', and '/products/lookupPrice' (circled with number 3). Each path item has a delete icon to its right.

Figure 5-19. Step 7: Review the API definition file with API Designer

**Step 7: Review the API definition file with API Designer.**

Review the draft APIs that your LoopBack application implements in the **API Designer** web application.

1. In the **Design** view of the **API Designer** web application, you can edit the API definition (or API interface) for your LoopBack application. You can also group a collection of APIs into an API product.
2. The categories column in the Design view corresponds to the sections of a OpenAPI document.
3. In this example, you define a **path** in your API. In a REST architecture, each API operation has two components: a path and an HTTP operation. In this example, the **GET /products/lookupPrice** API operation maps to the API that you developed in your LoopBack application.

## Step 8a: Review the data source in the API Designer

- Click **All APIs**
- Click the **Data Sources** tab

NAME	TYPE	
priceREST	LoopBack Data Source	

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

Figure 5-20. Step 8a: Review the data source in the API Designer

Click **All APIs** to go to the APIs tab in the Designer.

Click the **Data Sources** tab in the API Designer to see the data source that you generated.

The priceREST data source is displayed.

Click the data source to see the details.

You can review the data source definition from the Data Sources tab for the API in the API Designer.



## Step 8b: Test the data source connector in the API Designer

- Click the Test Data Source Connection icon

All Data Sources

Name: priceREST

Connector: REST services

Base URL: https://pricesample.mybluemix.net/price/

Options:

```
[{"template": {"method": "GET", "url": "https://pricesample.mybluemix.net/price/{id}", "headers": {"accepts": "application/json"}, "body": null}, {"template": {"method": "PUT", "url": "https://pricesample.mybluemix.net/price/{id}", "headers": {"accepts": "application/json"}, "body": {"id": 1, "name": "apple", "price": 100}}, {"template": {"method": "DELETE", "url": "https://pricesample.mybluemix.net/price/{id}", "headers": {"accepts": "application/json"}, "body": null}, {"template": {"method": "POST", "url": "https://pricesample.mybluemix.net/price", "headers": {"accepts": "application/json"}, "body": {"name": "apple", "price": 100}}]}
```

Operations:

Crud

Data source connection test succeeded  
a few seconds ago

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

Figure 5-21. Step 8b: Test the data source connector in the API Designer

When the data source detailed page is opened in the API Designer, click the Test Data Source Connection icon to test the connection.

In the example, the test ran successfully.

## Step 9: Start the LoopBack application and micro gateway

- To test your API, you must start two components:
  - The **LoopBack application** implements the API
  - The **micro gateway** makes the API available to consumers
- Start the LoopBack application and micro gateway in either of these ways:
  - From the terminal or command prompt:

```
$ apic start
```

- From the API Designer web application:

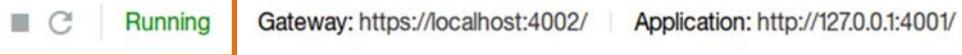


Figure 5-22. Step 9: Start the LoopBack application and micro gateway

Step 9: Start the LoopBack application and micro gateway.

To test your API, you must start two components:

- The **LoopBack application** implements the interaction API.
- The **micro gateway** makes the API available to consumers.

When you select start or restart in the API Designer, you implicitly run the `apic start` command. If you start the application and micro gateway with the `apic` command, you do not need to restart the application in the web application.



## Step 10: Test the API with the API Explorer

- Use the API Explorer as a test client for your interaction APIs

The screenshot shows the IBM API Connect interface. At the top, there's a navigation bar with 'IBM API Connect', 'pricing', 'Publish' (with a cloud icon), 'Explore' (which is highlighted with a red box), 'Settings', and a help icon. Below the navigation is a 'Drafts' section with a search bar and an 'Operations' dropdown set to 'by name'. A list of operations includes 'product.price', 'product.invoke', and 'product.lookupPrice' (which is selected and highlighted with a red box). To the right, there's a detailed view of the 'product.lookupPrice' operation. It shows security requirements: 'X-IB-M-Client-Id' located in the client header and 'X-IB-M-Client' located in the client secret header. Below this, there's a 'Definitions' section. On the far right, there's a 'GET https://localhost:4002/api/products/lookupPrice' section with 'Examples' (containing curl command 3) and a 'Try it' button (containing curl command 4). At the bottom, there are status indicators ('Running'), gateway information ('Gateway: https://localhost:4002/'), and application information ('Application: http://127.0.0.1:4001/').

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

Figure 5-23. Step 10: Test the API with the API Explorer

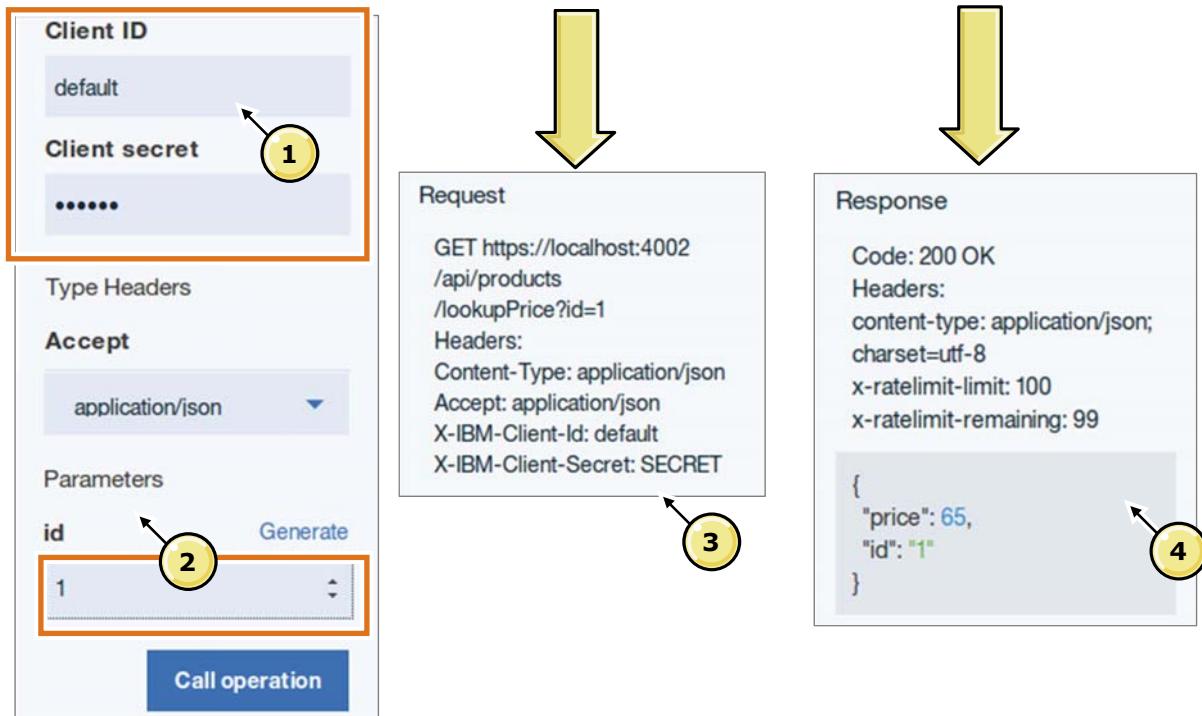
### Step 10: Test the API with the API Explorer.

The **API Explorer** is a view in the **API Designer** web application. The **API Explorer** is a test client for REST APIs that are hosted in your API gateway. You can use the API Explorer with the micro gateway or the DataPower gateway.

1. The leftmost column displays the API operations.
2. For each API operation, the details column displays the input requirements. In this example, the API definition expects `clientIdHeader` and `clientSecretHeader` fields in the HTTP request message header. The GET `products/lookupPrice` API operation has an optional parameter that represents the product identifier number.
3. The rightmost column displays sample source code that calls the API operation. In this example, the application code makes an HTTP GET request to the `/products/lookupPrice` web route.
4. Use the **Try it** option to test the operation with the required inputs.



## Example: Test API operations in Explorer



Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

Figure 5-24. Example: Test API operations in Explorer

### Example: Test API operations in Explorer.

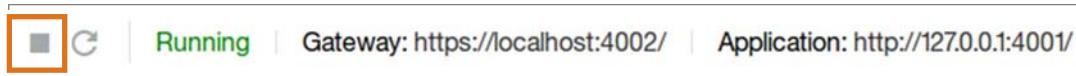
To view the API test client in the API Explorer view, scroll down in the rightmost column.

1. The client ID and client secret are auto-generated in the Explorer.
2. The **parameters** section lists the required and optional values that the API operation expects. In this example, you enter a product identifier with a value of 1. Optionally, you can **generate** random values based on the parameter data type.
3. Select **Call operation** to make an HTTP or HTTPS request to the API operation.
4. The response message header displays the HTTP status code and response header values.

The last section displays the HTTP response message body. In this example, the GET API operation returns a JSON object with the product price.

## Stop the gateway and application

- Click Stop the Servers icon in the API Designer



Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

Figure 5-25. Stop the gateway and application

After you are finished testing the application, click the Stop Servers icon to stop the gateway server and the application.

## 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 5-26. Relationship between API gateway and LoopBack

Considering the relationship between API gateway and LoopBack, why do you need to start two components?

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.

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

- Explain the concept of model-driven design
- Explain the purpose of the LoopBack Node.js framework
- Identify the components of a LoopBack application
- Create the application scaffold from the apic command-line utility
- Create a LoopBack application in the API Designer web application

## 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 Designer web application.
3. True or False: In IBM API Connect, a LoopBack application is a way to implement an API.



Figure 5-28. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

- True or False: You deploy your LoopBack application to the API Gateway.

The answer is False. Although you publish an endpoint for the API on the gateway, you deploy the LoopBack API application on a separate application server or container. The gateway does not host the LoopBack application.



- True or False: You must create a LoopBack application to edit the API interface definition in the API Designer web application.

The answer is False. You can create a OpenAPI definition in API Designer without creating a LoopBack application.

- 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: Creating a LoopBack application

Implementing APIs with the LoopBack framework

© Copyright IBM Corporation 2017

*Figure 5-30. Exercise: Creating a LoopBack application*

## Exercise objectives

- Create an API definition with the API Designer web application
- Create an application scaffold with the apic command-line utility
- Create LoopBack models and properties



Figure 5-31. 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 and properties in the API Designer
- Define models, properties, and relationships in the apic utility

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

© Copyright IBM Corporation 2017

*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 2017

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 2017

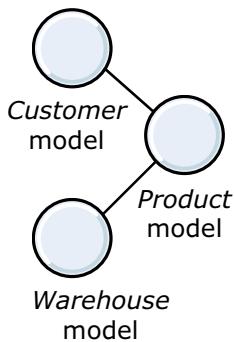
*Figure 6-3. Topics*

## LoopBack: An open source Node.js API framework

- LoopBack is an open source Node.js application framework for REST APIs
  - For more information, including documentation and source code on the LoopBack open source project, see: <http://loopback.io>
- 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 framework: Models, properties, relationships

- The **model** object represents the data and logic behind your API operations
- **Properties** represent a business data field
- **Relationships** define how API consumers create, retrieve, and modify models and model properties



LoopBack models, properties, and relationships

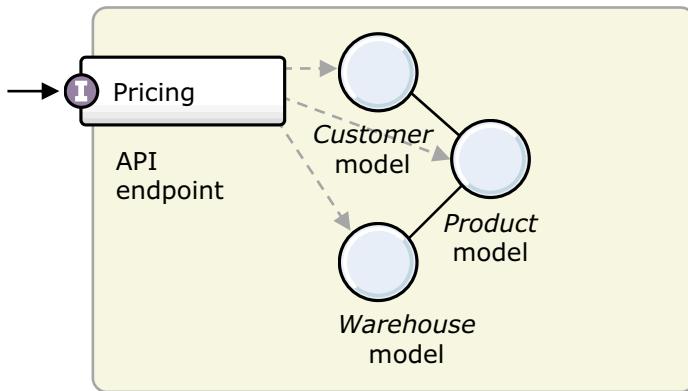
© Copyright IBM Corporation 2017

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

In this scenario, each of the models has a set of properties (not shown in the diagram). 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 2017

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

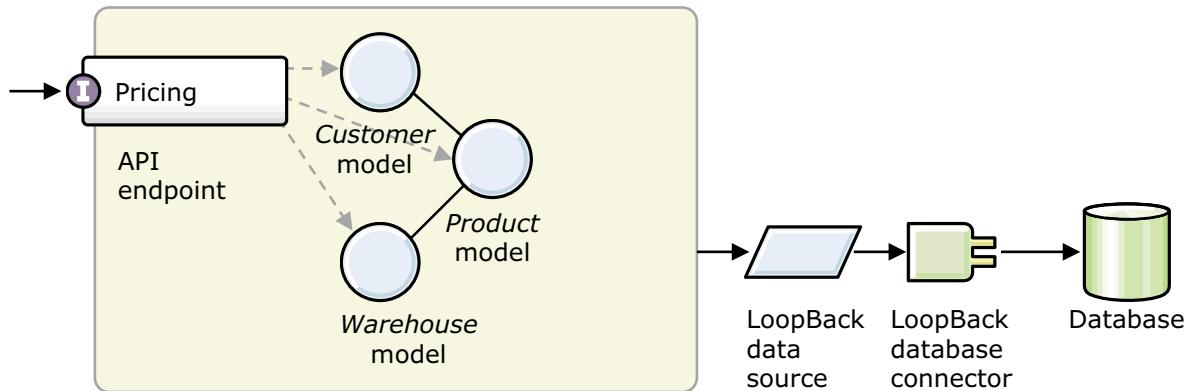
The LoopBack framework automatically adds create, retrieve, update, and delete API operations to each model object.

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 2017

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.

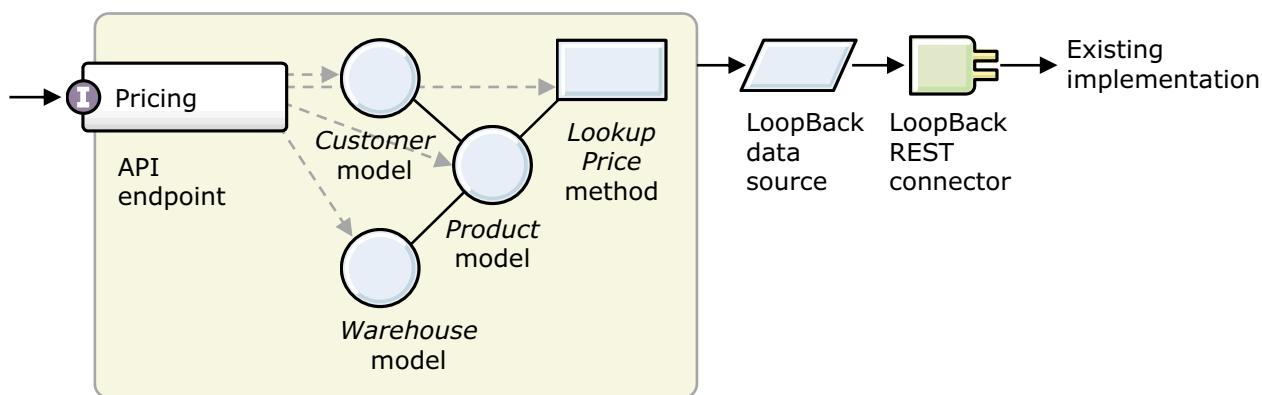
Non-database connectors do not support the persistence API: they call remote services and return data to a model object.

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: 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 2017

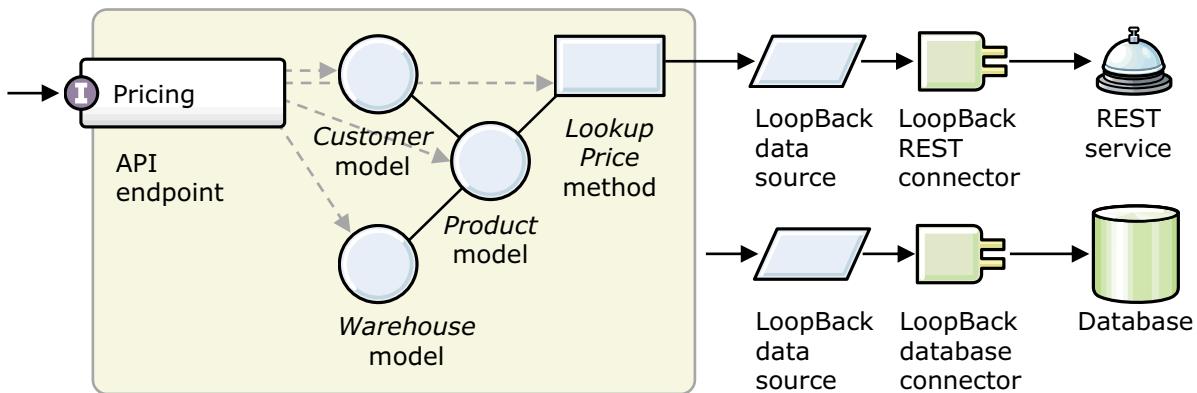
Figure 6-8. LoopBack framework: Remote methods and hooks

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 **GET /greet** request from the hello-world application. In this scenario, you can write a remote hook that saves the calculated price for a product after a client calls the *calculate price* API operation.

## 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



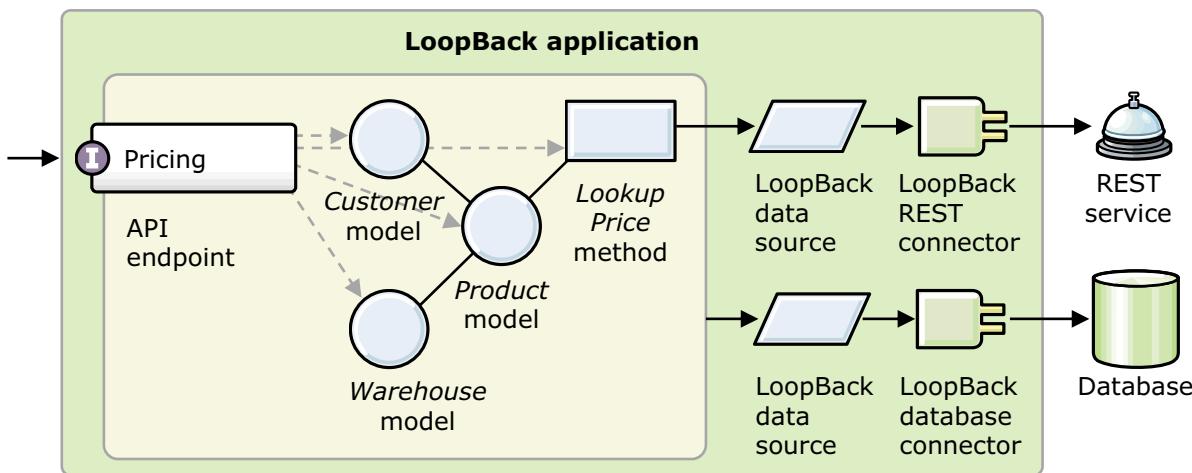
[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2017

Figure 6-9. LoopBack framework: Non-database connectors

## 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 2017

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 2017

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

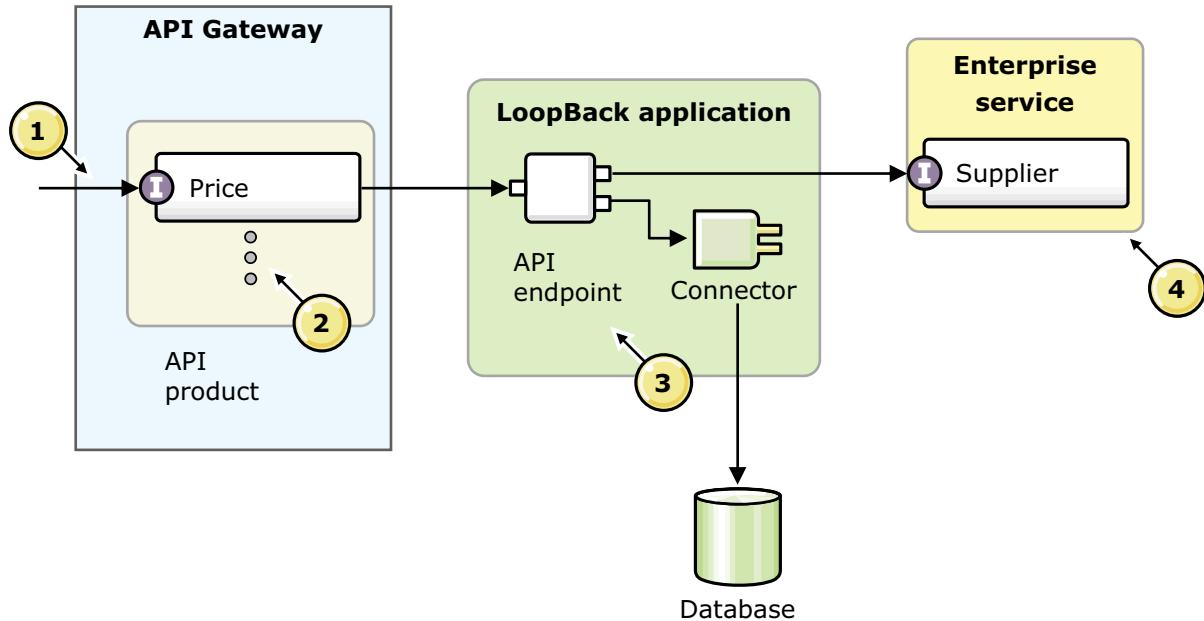
You configure two runtime components in an API Connect solution:

- 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 2017

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. This architecture has its own gateway and firewall to restrict access to clients within your organization.

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 **note** LoopBack sample application, the **note** model stores a greeting 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

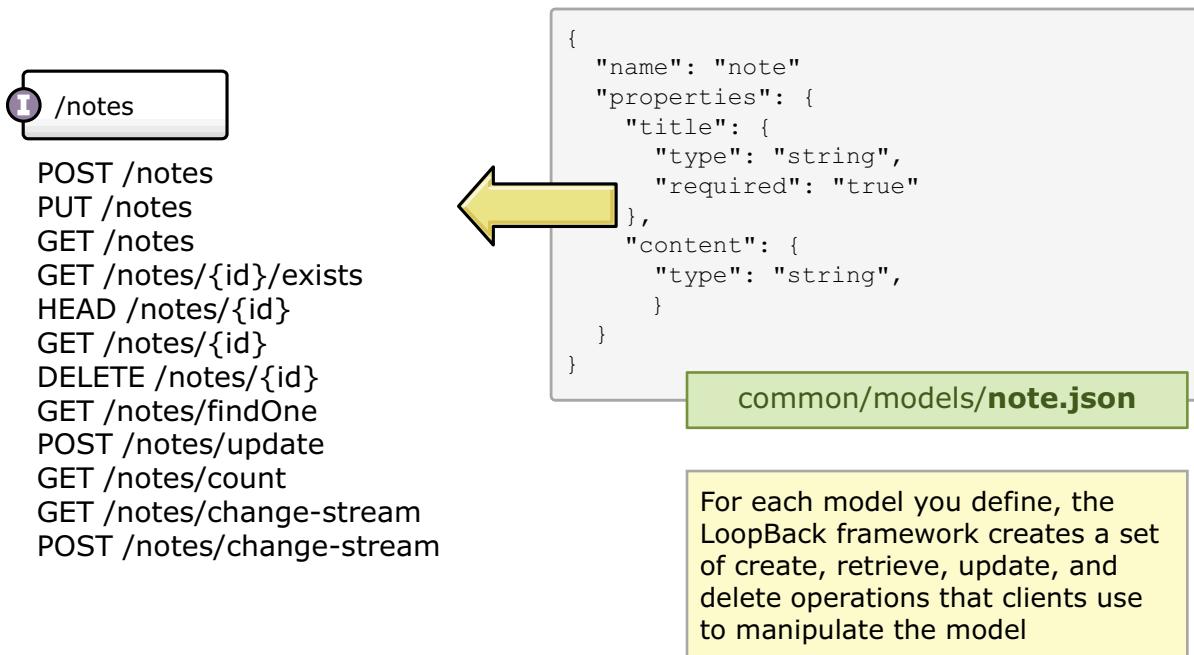
[LoopBack models, properties, and relationships](#)

© Copyright IBM Corporation 2017

*Figure 6-13. Example: Note sample application*

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

## Example: Note model



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

© Copyright IBM Corporation 2017

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.

## Example: Hello-world sample application

- The LoopBack framework maps the `greet` Node.js function to an API operation, `GET /messages/greet`
- When you call `GET /notes/greet?msg="IBM"`, the API operation returns `{"greeting": "Hello IBM"}` as a response message

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

© Copyright IBM Corporation 2017

*Figure 6-15. Example: Hello-world sample application*

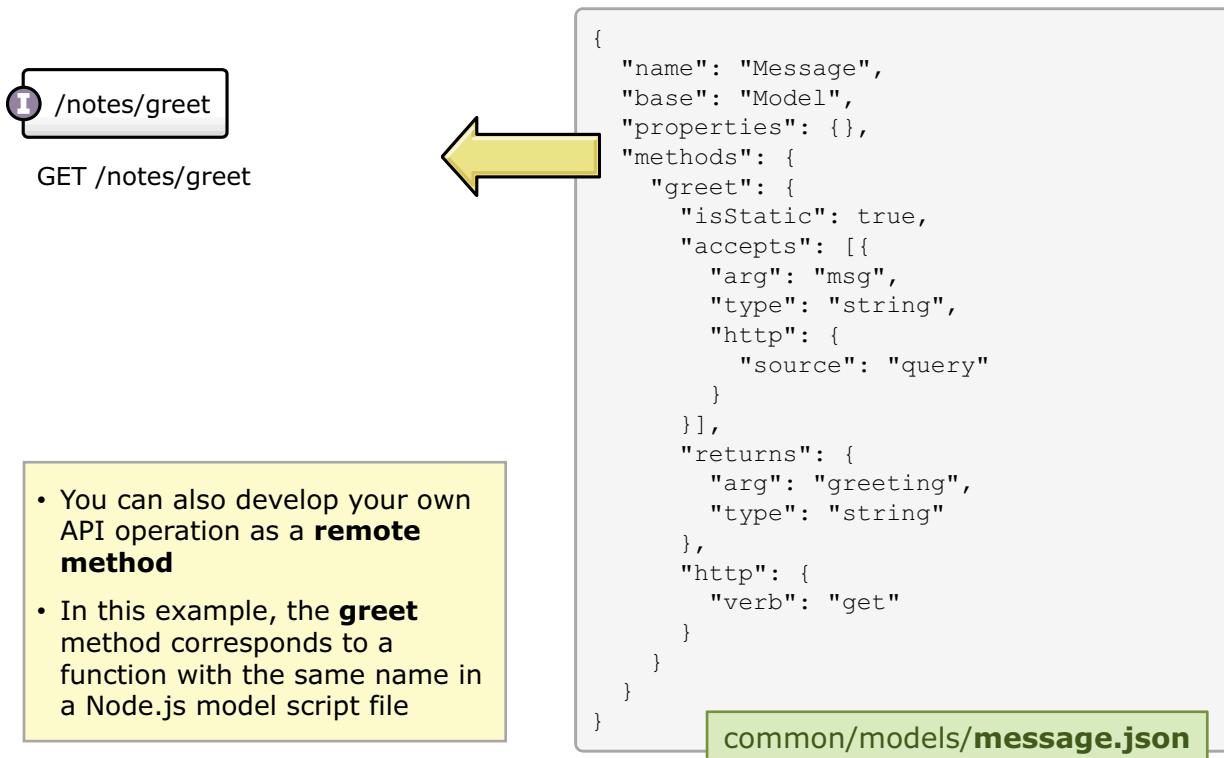
The second sample application in the “apic loopback” generator utility is the **Hello-world** sample application.

The LoopBack framework maps the `greet` Node.js function to an API operation, `GET /messages/greet`.

When you call `GET /notes/greet?msg="IBM"`, the API operation returns `{"greeting": "Hello IBM"}` as a response message.

As a static method, the `greet` function does not use the data from the properties in the Message model.

## Example: Hello-world message model



LoopBack models, properties, and relationships

© Copyright IBM Corporation 2017

Figure 6-16. Example: Hello-world message model

The **message.json** LoopBack model represents the contents of a message. When you call the `greet` method with a string input parameter, the API operation returns the string in a JSON object.

You can also develop your own API operation as a **remote method**. In this example, the `greet` method corresponds to a function with the same name in a Node.js model script file.

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

## Next steps after implementing the API

- After you developed and tested your API, you **deploy the LoopBack application** to a Node.js runtime environment
- You **define a security policy** in the API definition (interface) to secure client access to your API
- You **assemble message processing policies** at the API Gateway to mediate and control API access
- You **add the API definition** (interface) to a **product** in IBM API Connect to catalog the API
- You **publish the product** to the API Gateway to make it accessible to API consumers

## 6.2. LoopBack models, properties, and relationships

## LoopBack models, properties, and relationships

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2017

Figure 6-19. LoopBack models, properties, and relationships

## Topics

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

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

© Copyright IBM Corporation 2017

*Figure 6-20. 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. Define models, properties, and relationships**
- 3. Configure a data source to persist LoopBack model data**
- 4. Create remote methods and hooks to add custom business logic**
- 5. Review and test the API with the API Explorer web application**

Figure 6-21. Steps: LoopBack application, models, properties, relations

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

1. Generate a LoopBack application.
2. Define models, properties, and relationships.

## Generate a LoopBack application with apic loopback

- To implement an API, you create a LoopBack application with the **apic loopback** 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 three starting points for your LoopBack application:
  - 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 **note API** retrieves and saves its information into an in-memory data source

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

© Copyright IBM Corporation 2017

Figure 6-22. Generate a LoopBack application with apic loopback

## Example: Create a LoopBack application

```
$ apic loopback ← 1
? What's the name of your application? inventory ← 2
? Enter name of the directory to contain the project: inventory ← 3
? 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) ← 4
    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

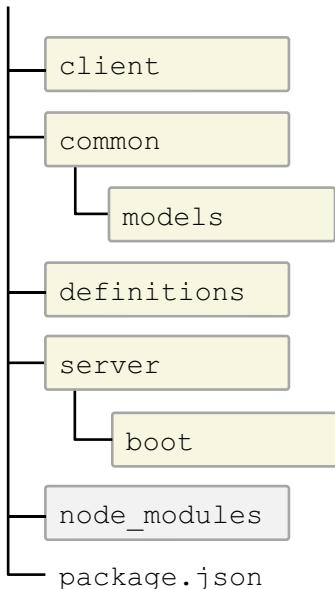
© Copyright IBM Corporation 2017

Figure 6-23. Example: Create a LoopBack application

1. In a terminal (Linux, Mac OS) or command prompt (Windows), run the `apic loopback` command. The **`apic`** 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
  - **definitions** stores the interface and product definition files for your API, in Swagger format
  - **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](#)

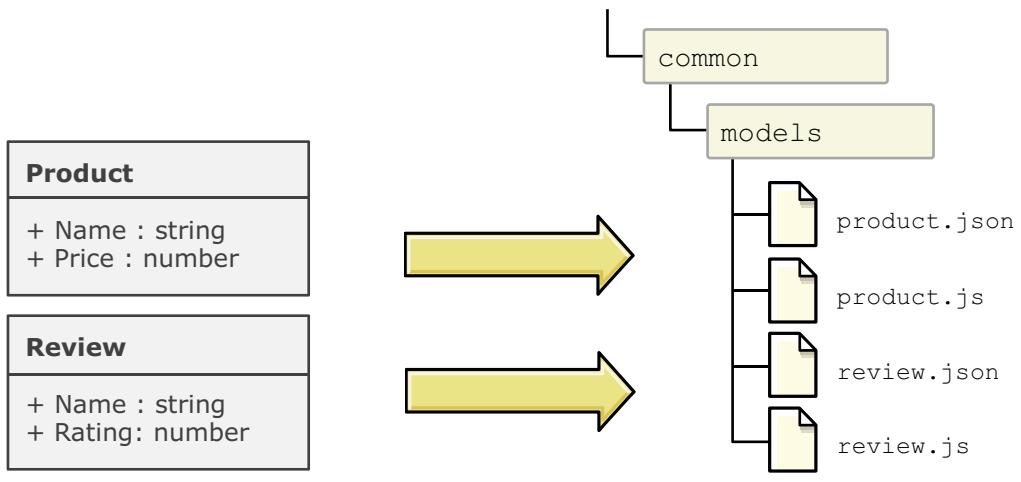
© Copyright IBM Corporation 2017

Figure 6-24. LoopBack application: Directory structure

The `node_modules` directory is a standard location to store application-specific Node modules. Therefore, it is not counted as one of the four directories in the LoopBack application structure.

## 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



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

© Copyright IBM Corporation 2017

Figure 6-25. 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 definition** file saves the characteristics of each model object. The model script defines custom logic for the model. Examples of custom logic include remote methods, validation functions, and operation hooks.

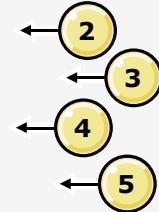
## Example: Create a model and properties

```
$ apic create --type model ← 1
? Enter the model name: product
? 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: 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:



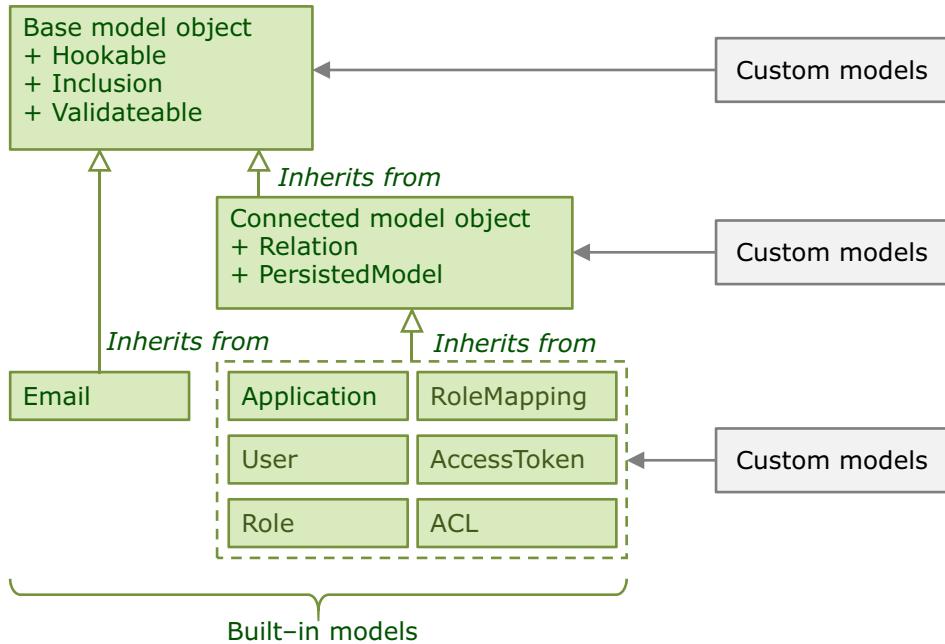
LoopBack models, properties, and relationships

© Copyright IBM Corporation 2017

Figure 6-26. Example: Create a model and properties

1. In a terminal (Linux, Mac OS) or command prompt (Windows), run the `apic create -type 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**.

## LoopBack model inheritance



## LoopBack models, properties, and relationships

© Copyright IBM Corporation 2017

*Figure 6-27. 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, 2017 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 2017

Figure 6-28. 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": {}
}
```

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

**common/models/product.json**

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

© Copyright IBM Corporation 2017

Figure 6-29. Example: Model definition file

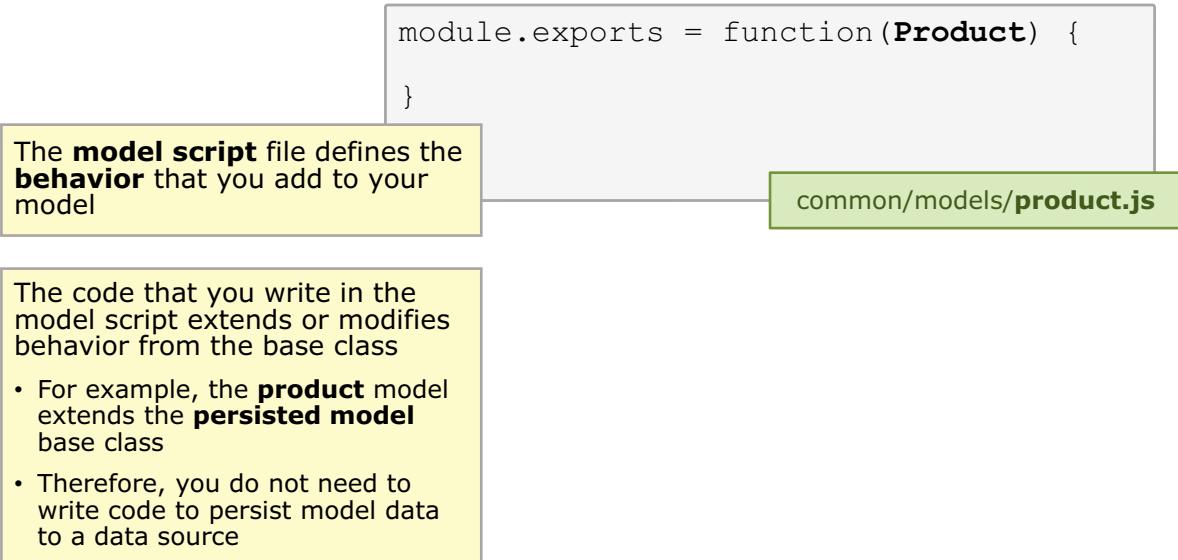
The **model definition** file captures the settings that you defined for your custom model. In this example, the name of the model is **product**. The LoopBack framework creates an instance of the base class, **PersistedModel**, as a template for your **product** model object.

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



*Figure 6-30. 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.

## 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 2017

Figure 6-31. 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 loopback: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**

LoopBack models, properties, and relationships

© Copyright IBM Corporation 2017

Figure 6-32. Example: Create model relations

1. In a terminal (Linux, Mac OS) or command prompt (Windows), run the `apic loopback: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 6-33. 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 <b>product</b> has many <b>reviews</b>
Belongs to	A <b>review</b> belongs to exactly one <b>product</b>
Has and belongs to many	A <b>warehouse</b> stores many <b>products</b> , and a <b>product</b> belongs to many <b>warehouses</b>
Has one	A <b>warehouse</b> has exactly one <b>product</b>
Has many through	A <b>warehouse</b> stores many products <ul style="list-style-type: none"> <li>▪ Create a third model (a through-model) that maps <b>warehouses</b> to <b>products</b></li> </ul>

- 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 2017

Figure 6-34. 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 and properties in the API Designer
- Define models, properties, and relationships in the apic utility

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

© Copyright IBM Corporation 2017

*Figure 6-35. 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 2017

Figure 6-36. Review questions

Write your answers here:

- 1.
  
- 2.

## 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 2017

Figure 6-37. 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 models and properties from a data source
- Build model instances from unstructured data

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-1. Unit objectives

## 7.1. LoopBack data sources

## LoopBack data sources

Defining data sources with connectors

© Copyright IBM Corporation 2017

*Figure 7-2. LoopBack data sources*

## Topics

### LoopBack data sources

- Generating LoopBack models from discovery and introspection

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-3. Topics

## Steps: LoopBack data sources

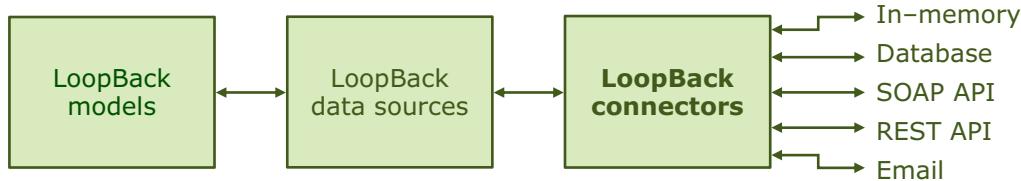
In this topic, you focus on the following LoopBack application implementation step:

1. Generate a LoopBack application
2. Define models, properties, and relationships
3. **Configure a data source** to persist LoopBack model data
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

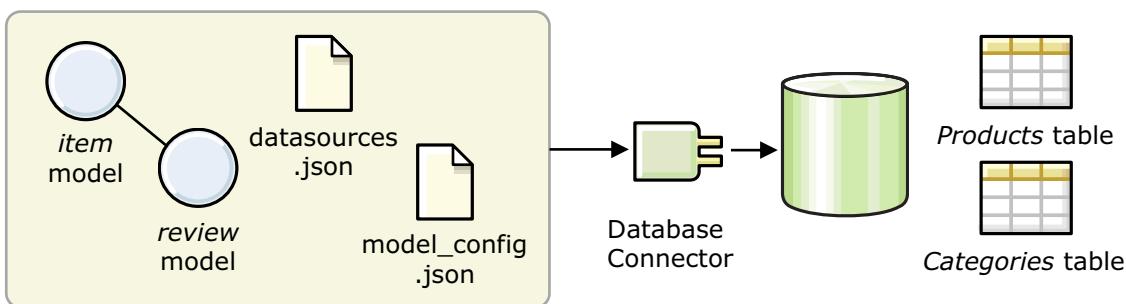
Defining data sources with connectors

© Copyright IBM Corporation 2017

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 2017

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
  - IBM Cloudant
  - DashDB
  - DB2, DB2 for iSeries, DB2 for z/OS
  - Informix
  - MongoDB
  - MySQL
  - Oracle
  - PostgreSQL
  - Redis
  - SQL Server
  - SQLite3
- 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 2017

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.

## Example: Create a memory data source

```
$ apic create --type 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
```

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-8. Example: Create a memory data source

1. To define a data source, run the `apic create -type 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 7-9. 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 2017

Figure 7-10. 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.

## Database transactions

- A **transaction** is a sequence of data operations that execute as a single, logical unit of work
  - Many relational databases support and coordinate transactions to enforce data consistency and business logic requirements
- A LoopBack model can perform operations in a transaction when you attach the model to one of the following connectors:
  - MySQL connector, with InnoDB as the storage engine
  - PostgreSQL connector
  - SQL Server connector
  - Oracle connector
- LoopBack transactions support the following isolation levels:
  - Transaction.SERIALIZABLE
  - Transaction.REPEATABLE\_READ
  - Transaction.READ\_COMMITTED (default setting)
  - Transaction.READ\_UNCOMMITTED

Figure 7-11. Database transactions

The purpose of database transactions is to run a set of data operations as one unit of work. The ability of a LoopBack application to set and use database transactions depends on the connector implementation and the database type. A LoopBack model can do operations in a transaction when you attach the model to one of the connectors that are listed on this slide.

## Reference: Transaction isolation levels

Level	Explanation
Serializable	<ul style="list-style-type: none"> <li>The database obtains read, write, and range locks</li> <li>The level is the highest transaction isolation level</li> </ul>
Repeatable reads	<ul style="list-style-type: none"> <li>The database obtains read and write locks, but not range locks</li> <li>The client can experience a <i>phantom read</i>, in which two identical queries return different collections of rows</li> </ul>
Read committed ( <i>default isolation level with LoopBack</i> )	<ul style="list-style-type: none"> <li>The database obtains read and write locks             <ul style="list-style-type: none"> <li>However, it releases read locks after it runs a <b>SELECT</b> statement</li> </ul> </li> <li>In addition to <i>phantom reads</i>, the client can also experience <i>non-repeatable read</i> <ul style="list-style-type: none"> <li>In this scenario, two queries within the same transaction return different values for the same row</li> </ul> </li> </ul>
Read uncommitted	<ul style="list-style-type: none"> <li>The level is the lowest transaction isolation level</li> <li>The client can experience <i>phantom reads</i>, <i>non-repeatable reads</i>, and <i>dirty reads</i> <ul style="list-style-type: none"> <li>In this scenario, the client reads modified but uncommitted changes</li> </ul> </li> </ul>

© Copyright IBM Corporation 2017

Figure 7-12. Reference: Transaction isolation levels

The SQL standard defines four isolation levels:

- **Serializable** is the highest isolation level. The database obtains read and write locks, and range locks.
- **Repeatable reads** is the second-highest isolation level. The database obtains read and write locks but not range locks. It is susceptible to phantom reads.
- **Read committed** is the default isolation level in LoopBack. The database obtains write locks and read locks. However, it releases read locks as soon as it runs a **SELECT** statement. It is susceptible to phantom reads and non-repeatable reads.
- **Read uncommitted** is the lowest isolation level. It's dirty reads, and it is susceptible to phantom reads and non-repeatable reads.

A transaction can use a *dirty read*, or an uncommitted read, to read data that was modified from another transaction, but not committed.

In a *non-repeatable read*, two queries within the same transaction return different values for the same row. If another transaction modifies a row before the database runs the second query, this phenomenon occurs.

The phenomenon of *phantom reads* can happen when a transaction that runs two identical queries receives different collections of rows. If another transaction inserts a row before the database runs the second query, this phenomenon occurs.

## 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: model classes that have 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 2017

Figure 7-13. 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 LoopBack models from discovery and introspection

## Generating LoopBack models from discovery and introspection

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-14. Generating LoopBack models from discovery and introspection

## Topics

- LoopBack data sources

 Generating LoopBack models from discovery and introspection

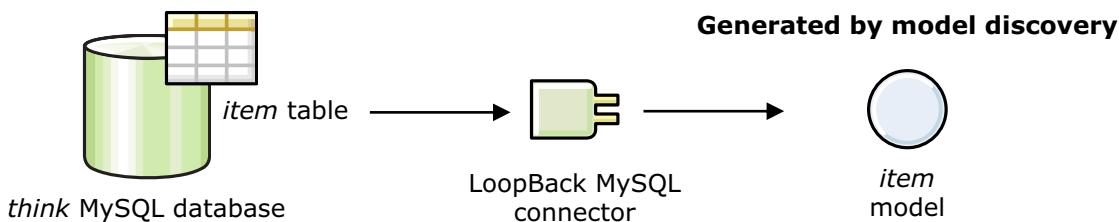
Defining data sources with connectors

© Copyright IBM Corporation 2017

*Figure 7-15. Topics*

## Model discovery and introspection

- You learned how to define LoopBack models and properties that correspond to a database schema
  - However, defining properties in the API Designer 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 2017

Figure 7-16. Model discovery and introspection

In the exercise case study, you defined the LoopBack model and properties with the API Design graphical editor. 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 four data sources support model discovery:
  - MySQL connector
  - PostgreSQL connector
  - Oracle connector
  - SQL server connector

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-17. 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 four data sources support model discovery:

- MySQL connector
- PostgreSQL connector
- Oracle connector
- SQL server connector

## API Designer: Discover Models and Update Schema

- The API Designer data source editor provides 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



- Notes:
  - If a model with the same name exists, you cannot generate a model with the **Discover Models** feature
  - Understand the effects of the **Update Schema** command: you are changing the database schema based on the models in your project

Figure 7-18. API Designer: Discover Models and Update Schema

The discover models feature has two conditions. If a model with the same name exists, you cannot generate a model with that name. You must also understand the effects of the Update Schema command – you are changing the database schema based on the models in your project.



## API Designer: Select properties to generate

- The **Discover Models** wizard retrieves a list of tables from the data source
- You select which tables you want to generate as a LoopBack model

Discover Models from: server.mysql-connection

Select the tables you want to generate models for

Search Table Names		Schema	selected properties	
<input checked="" type="checkbox"/>	item	think	7 selected properties	<b>Select Properties</b>

[Cancel](#) **Generate**

### Select Properties: Item

Select the Properties to Review. Required fields are automatically selected and can not be unchecked

Property Name	Type	Required	ID
<input checked="" type="checkbox"/> id	Number	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> name	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> description	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> img	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> imgAlt	String	<input type="checkbox"/>	<input type="checkbox"/>

Defining data sources with connectors

© Copyright IBM Corporation 2017

- For each table, select which columns you want to save as a LoopBack model property
- You must generate properties for required columns: you cannot clear these entries

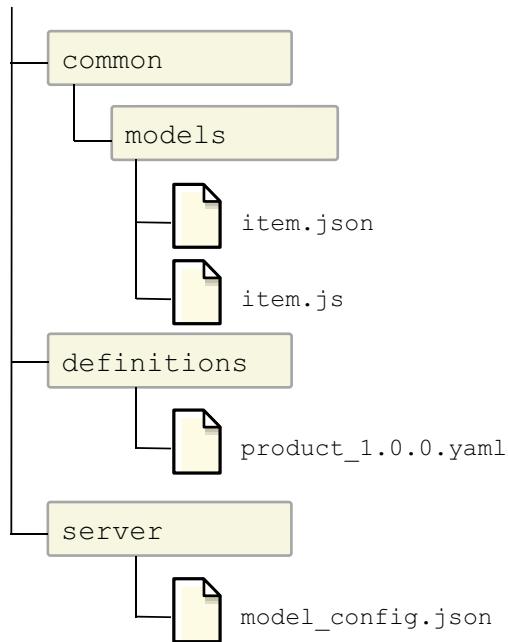
Figure 7-19. API Designer: Select properties to generate

After you select the **discover models** option in the data source toolbar, the wizard scans through the tables in the database. You can select one or more tables to generate as LoopBack models.

You can also select which columns within the table to save as LoopBack model properties. If the column is optional, you have the choice of omitting the property from the model. For required columns, you must generate a corresponding property in the model.

When you click **Generate**, the wizard creates the **model configuration file** and **model script file**, which define the model. API Designer saves these files in the model directory of your API application. It also updates the API definition with the paths and operations to access the LoopBack models.

## Model discovery: What files does it generate?



For each database table that you select, the **discover models** wizard generates the **model definition file** and **model script file** that defines the model data structure

The wizard updates the **API definition** file with the **paths**, **operations**, and **request and response message types** that map to the LoopBack model API

The wizard adds one entry in the **model configuration file** to **register the LoopBack model** with the **data source**

Figure 7-20. Model discovery: What files does it generate?

## Example: Review the item MySQL database table

```
$ mysqlshow think item

Database: think  Table: item
+-----+-----+-----+-----+-----+
| Field | Type   | Collation | Null | Key |
+-----+-----+-----+-----+-----+
| id    | int(11) | latin1_swedish_ci | NO  | PRI |
| name  | varchar(45) | latin1_swedish_ci | NO  |       |
| description | varchar(2000) | latin1_swedish_ci | NO  |       |
| img   | varchar(255) | latin1_swedish_ci | NO  |       |
| img_alt | varchar(45) | latin1_swedish_ci | YES |       |
| price | varchar(45) | latin1_swedish_ci | NO  |       |
| rating | varchar(45) | latin1_swedish_ci | YES |       |
+-----+-----+-----+-----+-----+
```

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-21. Example: Review the item MySQL database table

The **mysqlshow** utility displays the database schema for the **item** table in the **think** database. This utility is part of the MySQL database installation; it is not part of the IBM API Connect Toolkit.

The database has six columns, not including the primary key “**id**” identifier field. The **null** setting indicates whether a particular field accepts *null* values in its data. Fields that do not permit null values are **required** fields.

The **varchar** type indicates that the six columns accept string data. The **id** field is an integer value.

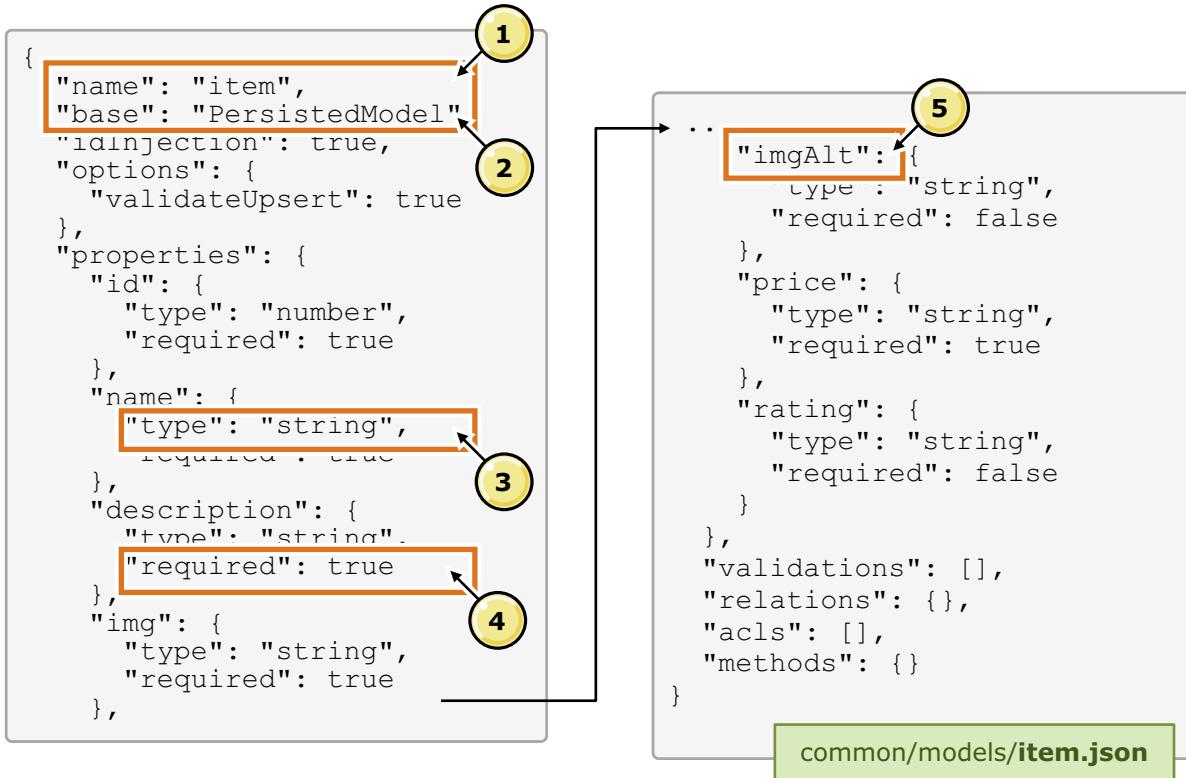
Not shown in the screen capture are the permitted SQL operations and the composition of the **id** primary key field. The MySQL database automatically generates a unique value in the **id** field when a client inserts a row into the **item** table.



### Syntax

```
mysqlshow --user=student --password=Passw0rd! think item
```

## Example: Generated item.json model definition file



Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-22. Example: Generated item.json model definition file

The following chart displays the generated model definition file from the `item` relational database table.

1. The name of the LoopBack model class is the name of the database table: `item`.
2. The `PersistedModel` base class provides a set of Node functions that create, retrieve, update, and delete records from the `item` database. The LoopBack framework also generates a set of REST operations that correspond to these data-centric model functions.
3. The properties and property data types correspond to the column data types in the `item` table. Each data source connector defines a mapping between JSON data types and database data types. For example, the `string` data type holds a `varchar` value from the database table.
4. Any MySQL database fields that do not accept `null` values become `required` values in the corresponding LoopBack model.
5. The model discovery feature converts column names in lowercase dashed names (for example, `img_alt`) to camel case spelling (for example, `imgAlt`). The data source connector automatically converts between the different spellings of the properties.

## 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:
  - Memory data source
  - MongoDB data sources
  - REST data sources
  - SOAP data sources

Figure 7-23. 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: 'warrenf',
      address: {
        street: '8200 Warden Ave',
        city: 'Markham',
        region: 'Ontario',
        postcode: 'L6G 1C7',
        country: 'Canada'
      },
      email: [
        {label: 'work', id: 'warrenf@example.com'}
      ],
      tags: ['ibm', 'cloud', 'apiconnect']
    }
  });
}
```

Defining data sources with connectors

© Copyright IBM Corporation 2017

*Figure 7-24. 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` directory, under `demos/introspection`.

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.

## 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 2017

*Figure 7-25. 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 models and properties from a data source
- Build model instances from unstructured data

Defining data sources with connectors

© Copyright IBM Corporation 2017

*Figure 7-26. 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 data source?
  - A. MySQL connector
  - B. Cloudant connector
  - C. MongoDB connector
  - D. Remote connector



Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-27. 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 data source?

- A. MySQL connector
- B. Cloudant connector
- C. MongoDB connector
- D. Remote connector

The answer is A.

## Exercise: Defining LoopBack data sources

Defining data sources with connectors

© Copyright IBM Corporation 2017

Figure 7-29. Exercise: Defining 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 API Explorer



Defining data sources with connectors

© Copyright IBM Corporation 2017

*Figure 7-30. 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

## 8.1. LoopBack remote methods and hooks

## LoopBack remote methods and hooks

Implementing remote methods and event hooks

© Copyright IBM Corporation 2017

*Figure 8-2. LoopBack remote methods and hooks*

## Topics

- ▶ LoopBack remote methods and hooks

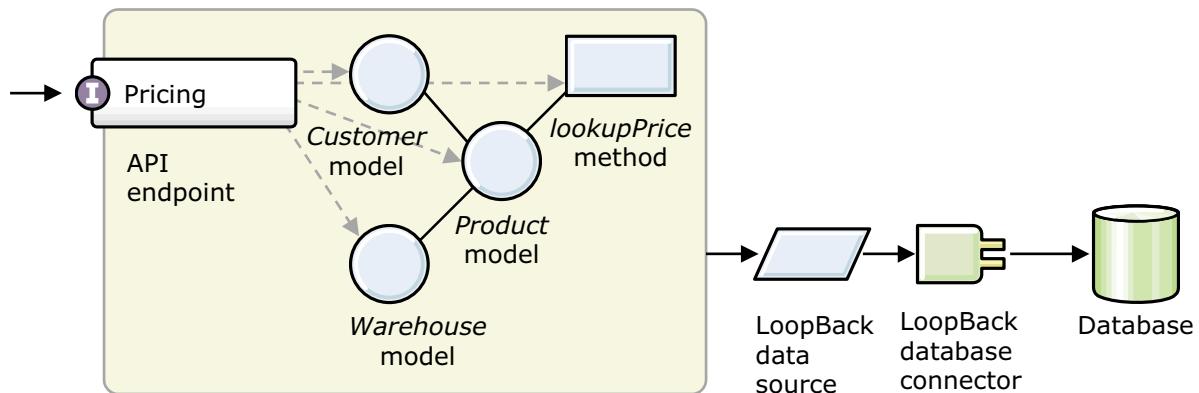
Implementing remote methods and event hooks

© Copyright IBM Corporation 2017

Figure 8-3. Topics

## 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 2017

Figure 8-4. 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 **GET /greet** request from the hello-world 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. Define models, properties, and relationships
3. Configure a data source to persist LoopBack model data
4. **Add remote methods and hooks** to add custom business logic
5. Review and test the API with the API Explorer web application

Figure 8-5. 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

## 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 2017

Figure 8-7. 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 2017

Figure 8-8. 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 and sanitize 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

## 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 2017

Figure 8-10. Example: Remote hook

This remote hook example is taken from the *remote* exercise. 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>

## 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"> <li>Transform data from a data source before LoopBack creates a model           <ul style="list-style-type: none"> <li>For example, decrypt data for model properties</li> </ul> </li> </ul>
persist	Triggered before LoopBack persists model data to a data source	<ul style="list-style-type: none"> <li>Transform data before storing in a data source           <ul style="list-style-type: none"> <li>For example, encrypt data from model</li> </ul> </li> </ul>

Implementing remote methods and event hooks

© Copyright IBM Corporation 2017

Figure 8-12. 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 8-13. 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 2017

*Figure 8-14. 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 2017

*Figure 8-15. Example: After save operation hook*

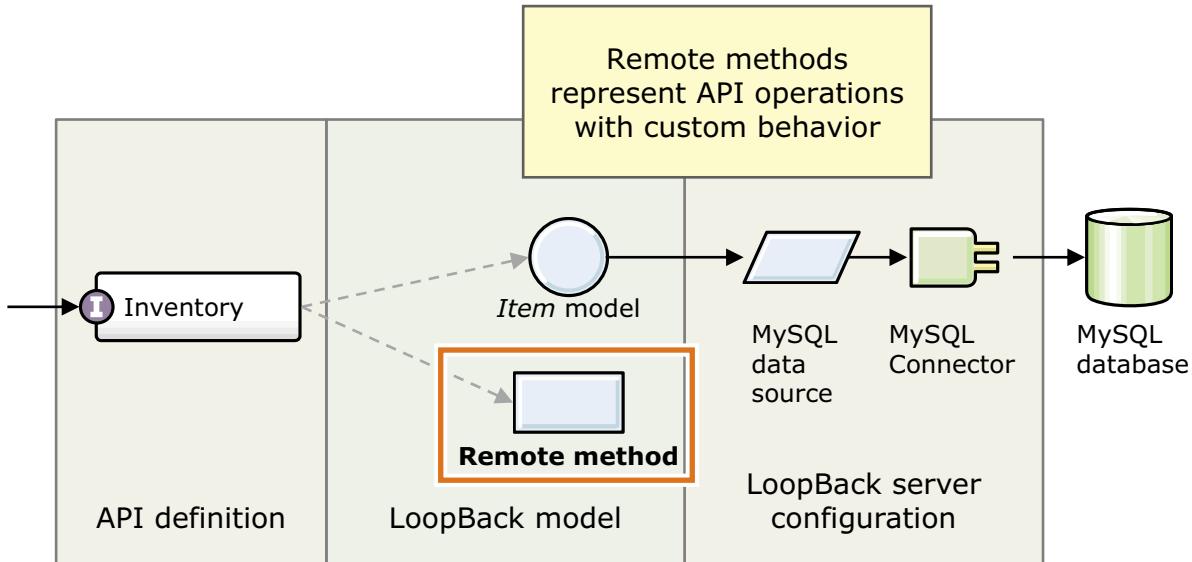
In this example, the `Item.observe` function creates an operation hook for the **after save** event. The `logCreateUpdate` function checks whether the LoopBack framework created or updated the **item** model object. If the framework created an instance of **item**, the `ctx.instance` property is a reference 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 2017

Figure 8-16. When do you implement remote methods?

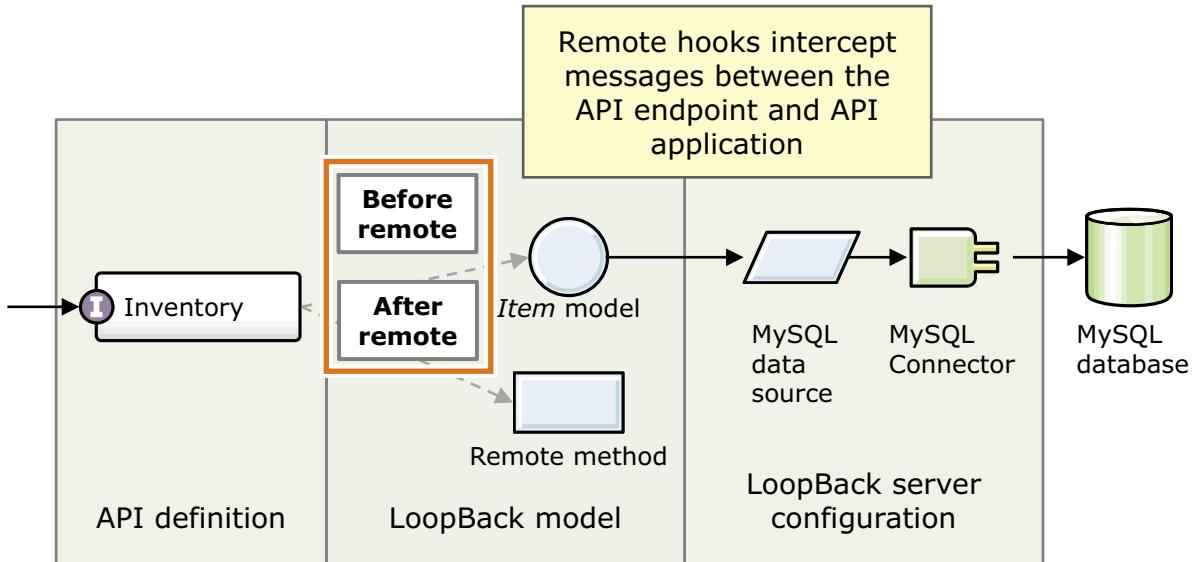
The LoopBack API application that you create has three logical components:

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

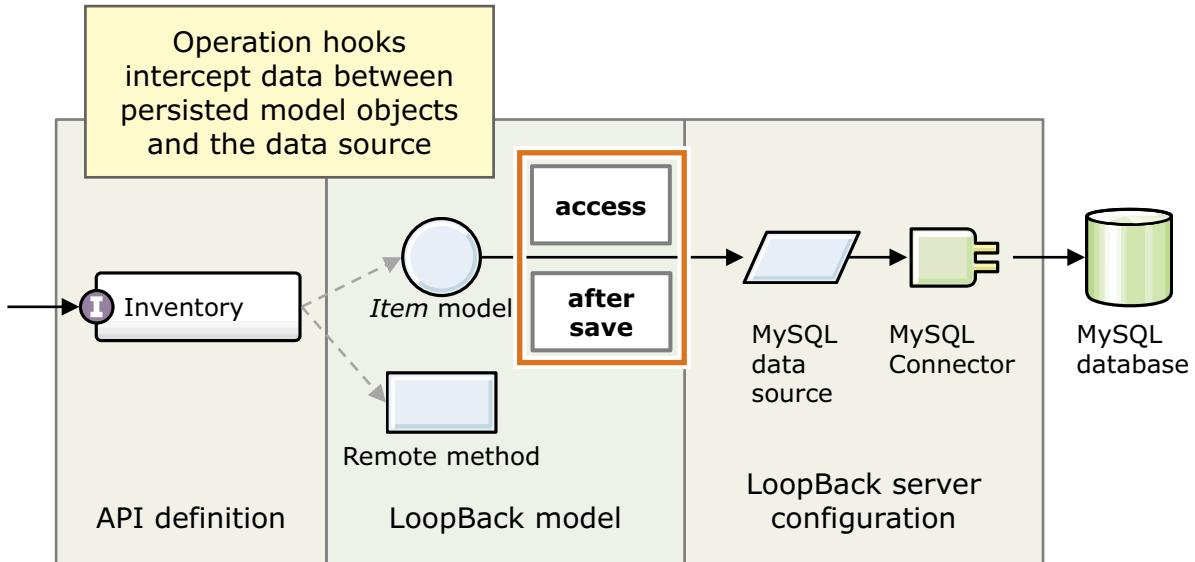
© Copyright IBM Corporation 2017

Figure 8-17. 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 2017

Figure 8-18. 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 2017

*Figure 8-19. 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 2017

Figure 8-20. 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 2017

Figure 8-21. Review answers

## Exercise: Implementing event-driven functions with remote and operation hooks

Implementing remote methods and event hooks

© Copyright IBM Corporation 2017

Figure 8-22. Exercise: Implementing event-driven functions with remote and operation hooks

## Exercise objectives

- Define a remote hook in a LoopBack application
- Define an operation hook in a LoopBack application
- Test remote and operation hooks in the assembly editor test client



---

# Unit 9. Assembling message processing policies

## Estimated time

01:30

## 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 Designer.

## 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
- List the policies that each API gateway type supports
- Explain how to generate JSON Web Tokens with policies
- Explain the concept and use case for custom user-created policies

## 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 2017

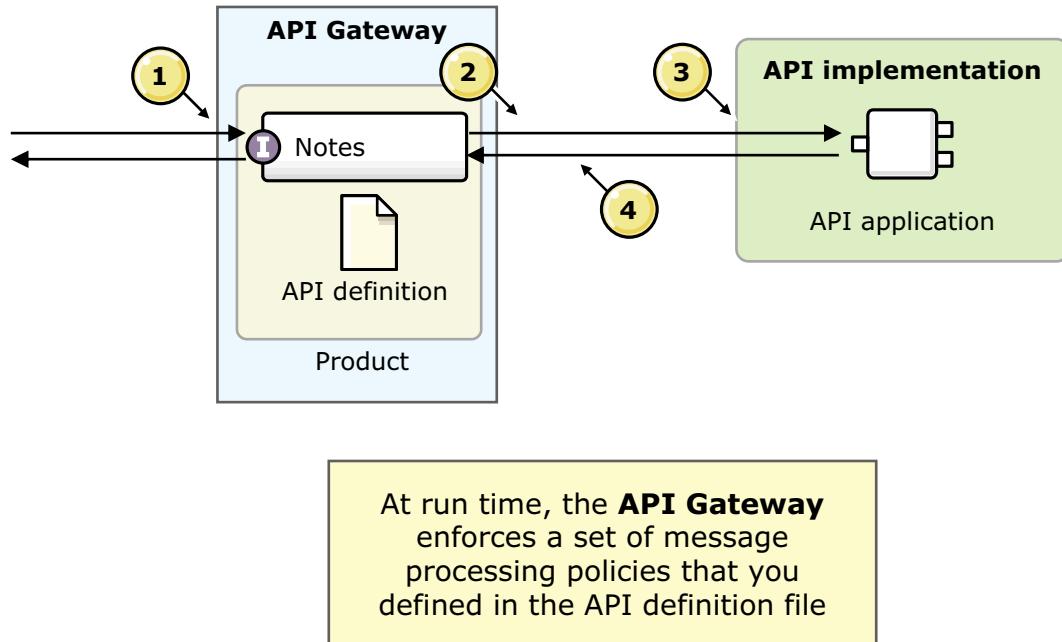
Figure 9-2. What is a message processing policy?

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 Designer application and the API Manager refer 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



Assembling message processing policies

© Copyright IBM Corporation 2017

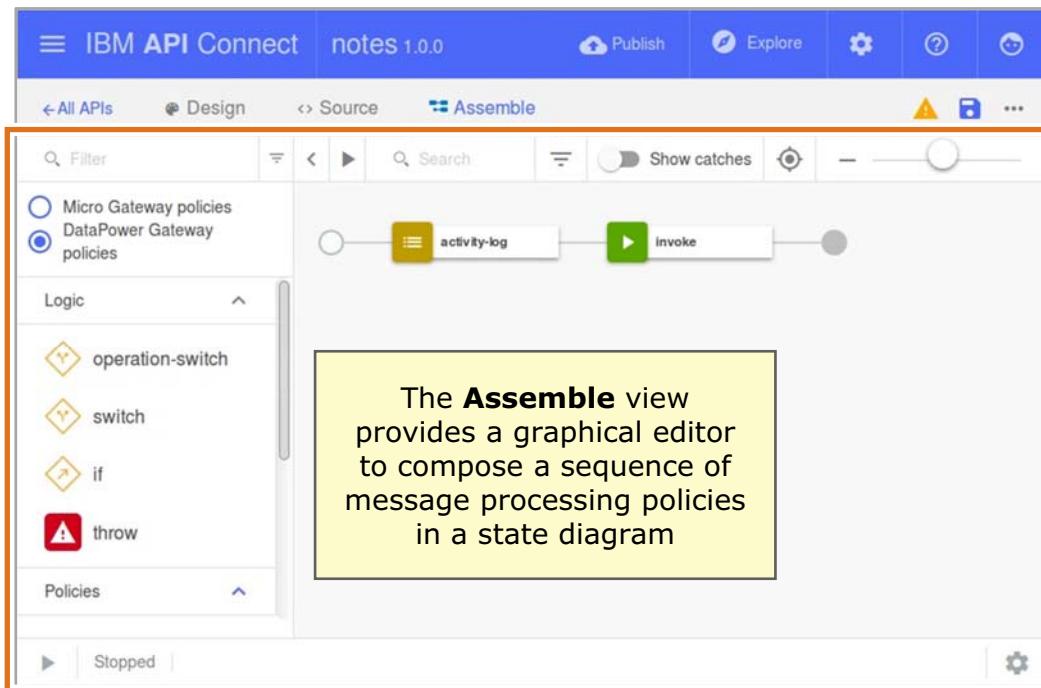
Figure 9-3. Message processing policies at run time

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 sending the HTTP response message back to the client application.



## Assembly editor: Creating policy assemblies



Assembling message processing policies

© Copyright IBM Corporation 2017

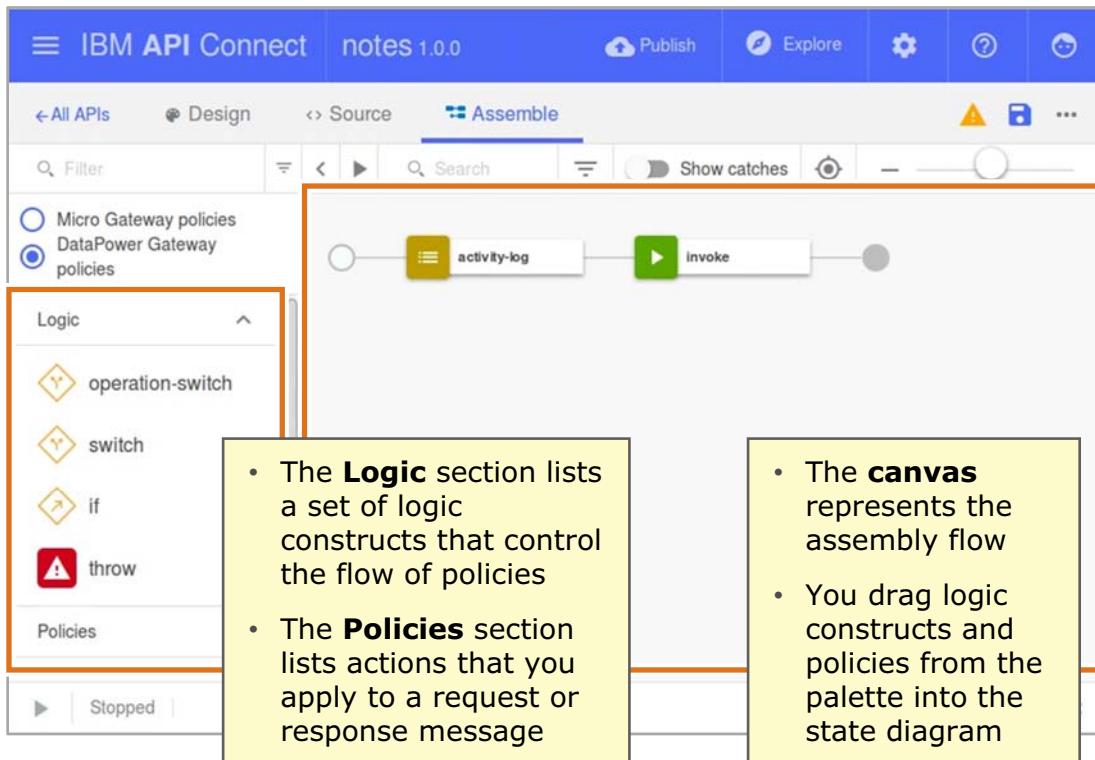
Figure 9-4. Assembly editor: Creating policy assemblies

The **Assemble** view in the API Designer 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 Designer application writes the policies and logic constructs in the **assemble** section of the **x-ibm-configuration** extension entry of the Swagger (OpenAPI) document when you save the API definition file.



## Assembly editor: Palette and canvas



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-5. Assembly editor: Palette and canvas

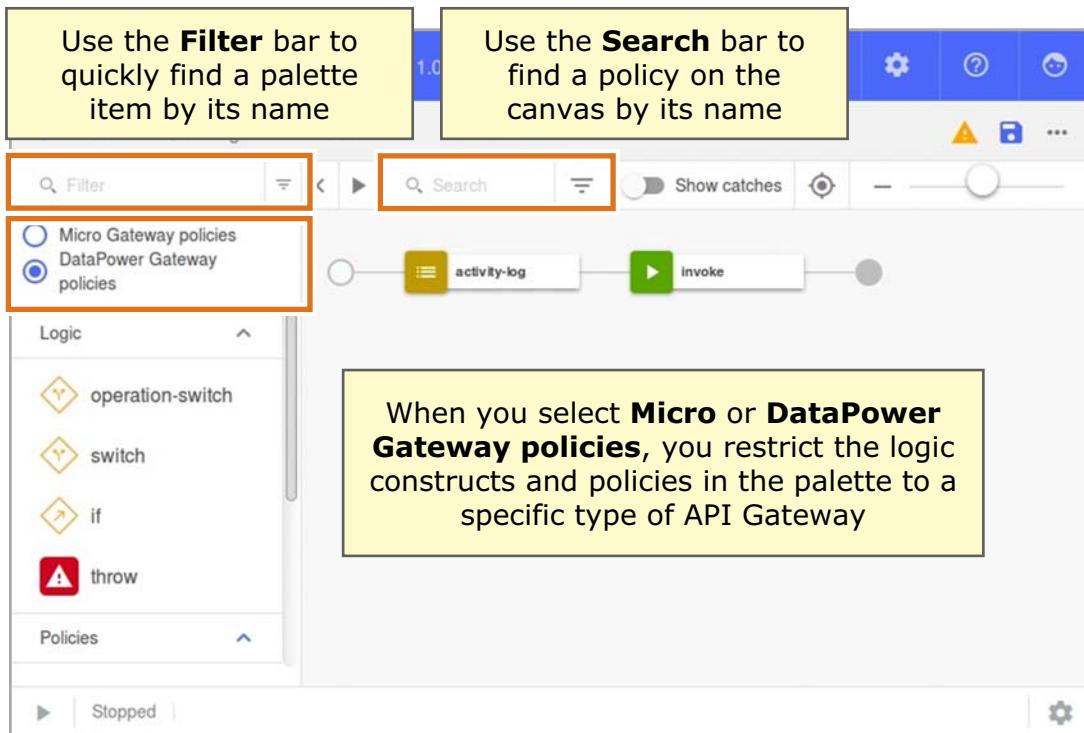
The **palette** lists the configuration constructions that you can place onto the **canvas**.

Logic constructs control the message flow across policies. For example, you can set an **operation-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 2017

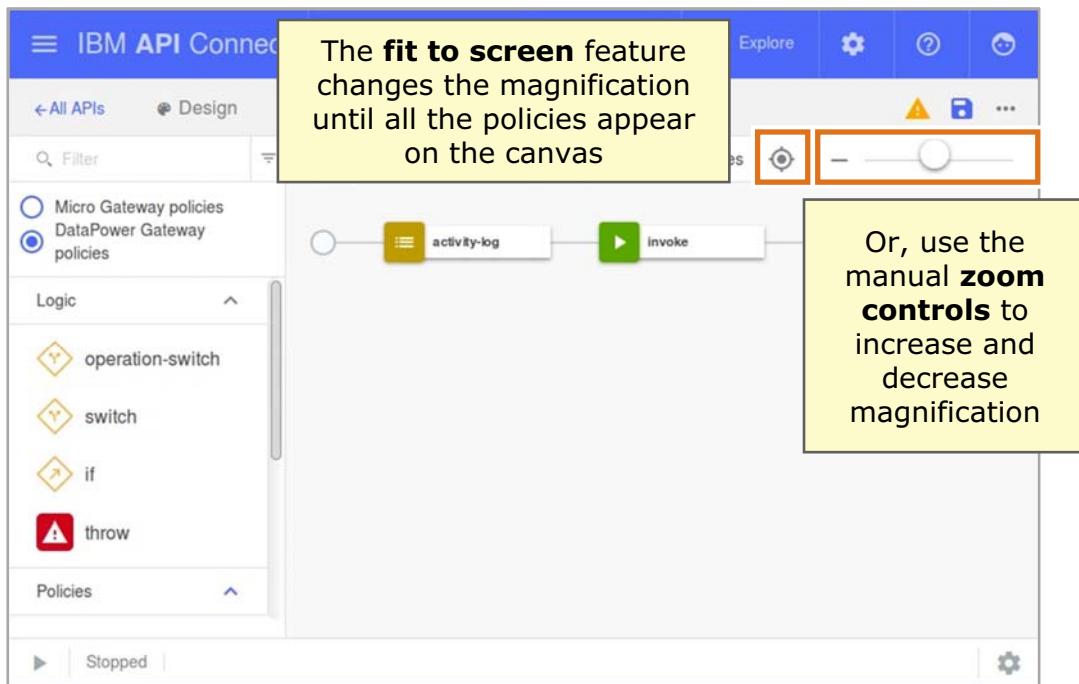
Figure 9-6. Assembly editor: Filter, search, and gateway type

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.

When you select the **Gateway Policies** radio button, you change the list of logic constructs and policies in the palette. More importantly, you define the target API gateway type. For example, you cannot deploy an API definition to a Micro Gateway after you set the **DataPower Gateway policies** option in the assemble view.



## Assembly editor: Magnify and zoom



Assembling message processing policies

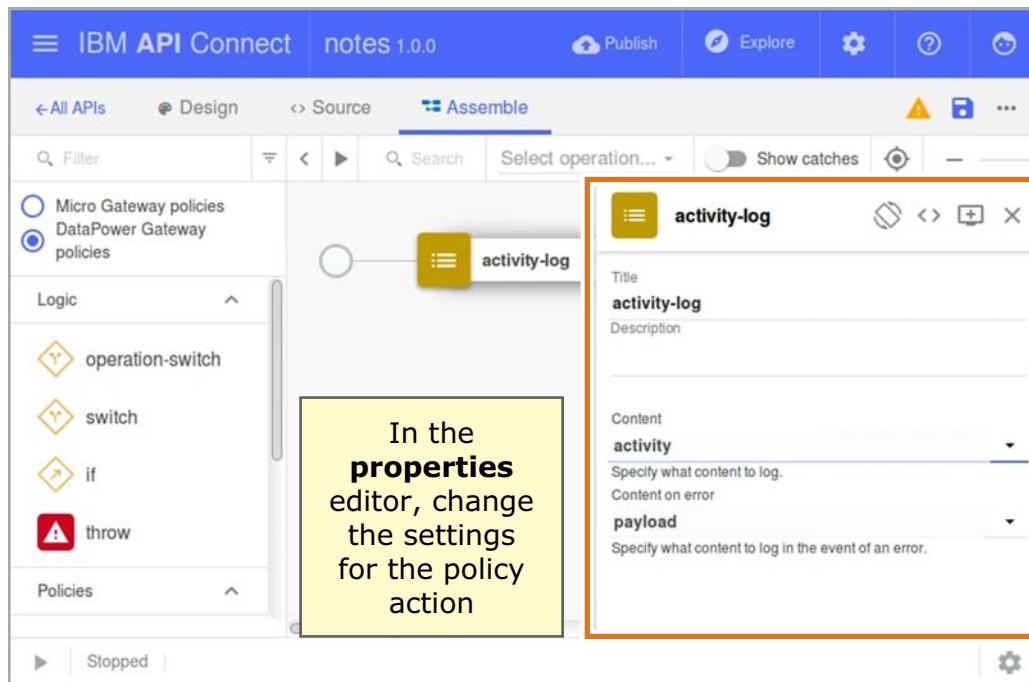
© Copyright IBM Corporation 2017

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 appears 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

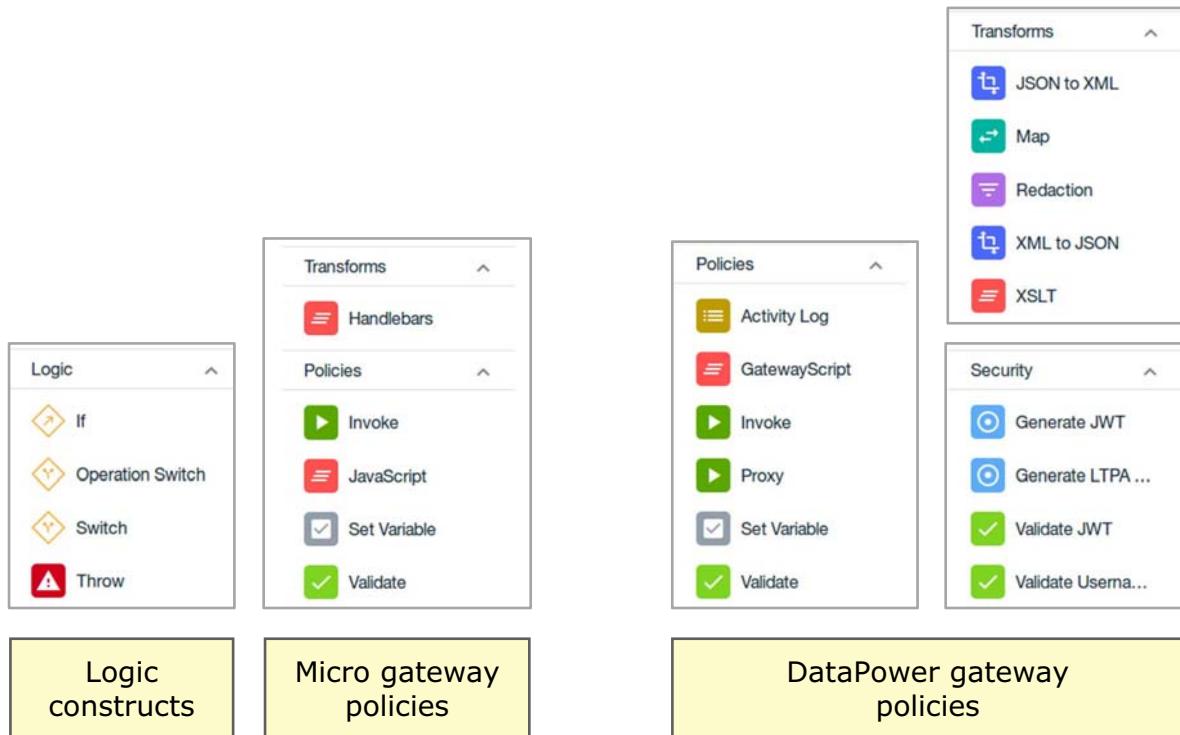
© Copyright IBM Corporation 2017

Figure 9-8. Assembly editor: Properties editor

The **properties** editor reveals more settings for the selected policy. In this example, the **activity-log** policy has a title, a description, and two settings. During a normal API request, the activity-log policy captures basic metadata on the path and parameters of the call. If an error occurs during processing, the activity-log captures the entire HTTP headers and message body. The **activity-log** policy is available only on DataPower Gateways.



## API policies and logic constructs



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-9. API policies and logic constructs

The **palette** in the **assembly editor** divides the list of items into two categories: **logic constructs** and **policies**.

**Logic constructs** change the sequence in which the gateway runs policy actions. Both the micro gateway and DataPower gateway support the **operation-switch**, **switch**, **if**, and **throw** constructs.

**Policies** perform an action on the HTTP message, or an environment variable. The **micro** gateway supports four policies: **validate**, **invoke**, **JavaScript**, and **set-variable**.

The **DataPower gateway** supports its own set of policies, transforms, and security policies.

Note: The micro gateway and DataPower gateway support slightly different types of JavaScript code. Therefore, the micro gateway runs JavaScript code in a **JavaScript** policy. The DataPower gateway runs JavaScript code in a **Gateway script** policy. These two policies are not interchangeable – you must use the correct policy according to 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 request message payload into API analytics
7. Generate and store a JSON Web Token in an API call

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.
- Save a copy of the request message payload into API analytics.
- Generate and store a JSON Web Token in an API call.

The following set of slides explain how to handle these scenarios with policies.

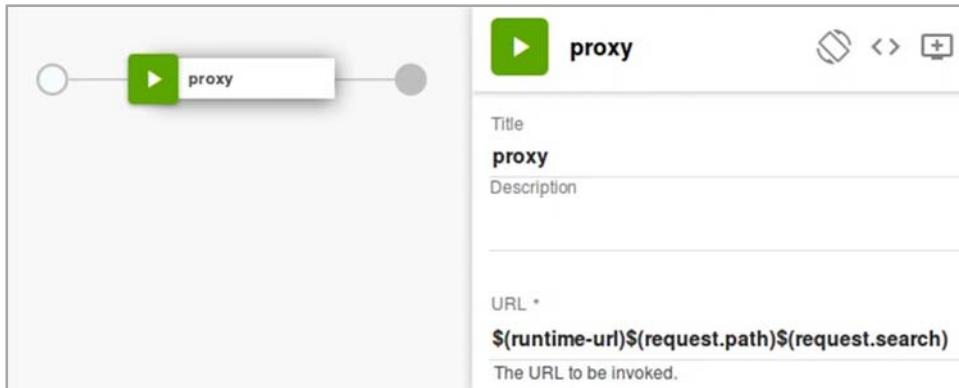
## Example one: Forward an API call with the invoke policy



- When you create an API definition, API Designer 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

Figure 9-11. Example one: Forward an API call with the invoke policy

## Example one: Forward an API call with the proxy policy



- **DataPower Gateways** support two policies to call remote services:
  - **Invoke** policy
  - **Proxy** policy
- If the remote API returns messages in a multi-part format, use the proxy policy
- If you want to call multiple services in a single assembly flow, use the invoke policy

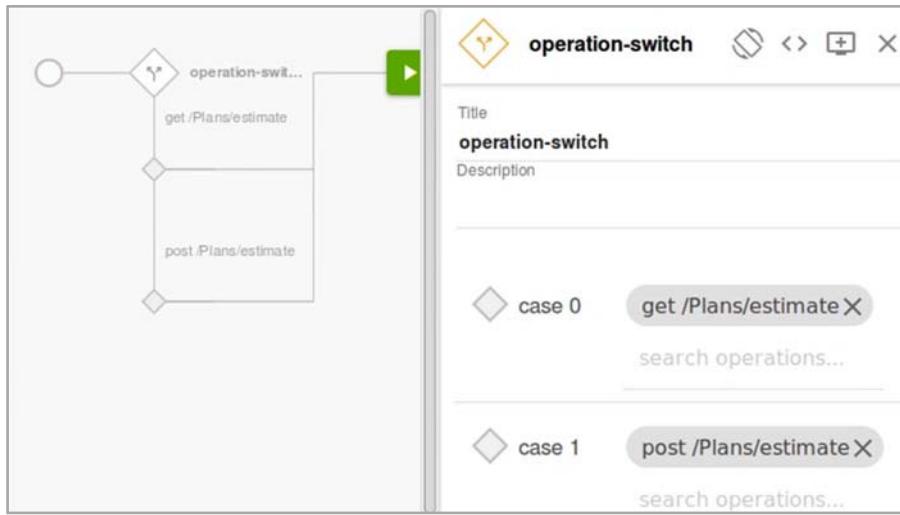
Figure 9-12. Example one: Forward an API call with the proxy policy

## What is the difference between invoke and proxy policies?

- Both the **invoke** and **proxy** policies send an HTTP request to a network endpoint
- The **invoke** policy is designed to call any arbitrary HTTP service
  - You can combine the output from multiple **invoke** calls into a single request message
- The **proxy** policy is designed to forward the current API operation to another API
  - When the Gateway runs through an assembly flow, it can run one policy only
- The **invoke** policy does not support multi-part messages
  - You must use a **proxy** policy at the end of an assembly flow to call an API that returns a multi-part message

## Example two: Switch case by API operation

- When the **operation-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 2017

Figure 9-14. Example two: Switch case by API operation

## 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 2017

Figure 9-15. Example three: Map multiple API calls into a response

## 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

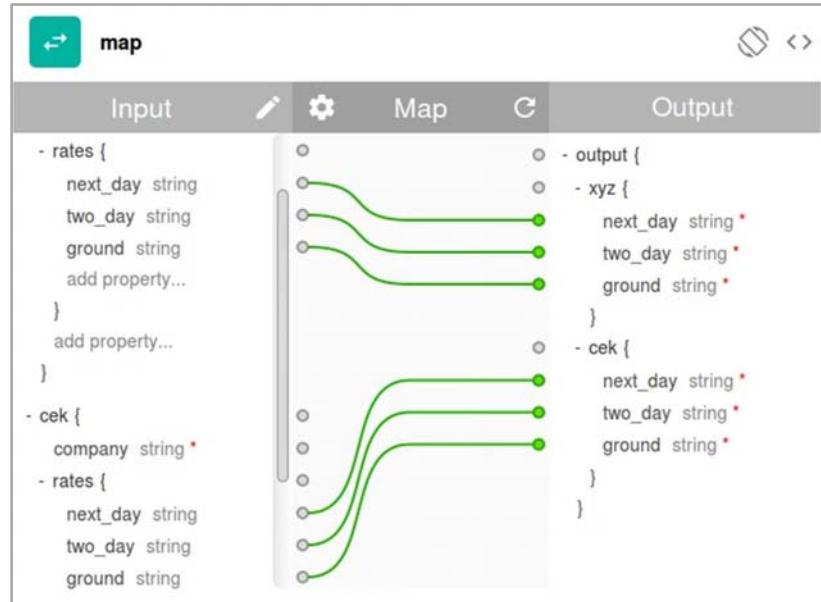
Content type	Definition *
application/json	Inline schema

[+ input](#)    [+ parameters for operation...](#)

Figure 9-16. Example: Define input parameters in a map policy

## Example: Map API results into a single response message

- In the **output** column, you define the structure of the message payload after the **map** policy
- In the **map** column, draw a **wire** between the nodes in the input and output columns to **copy** properties into the output message
- In this example, you aggregated shipping rates from two delivery companies into the API response message



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-17. Example: Map API results into a single response message

## 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 **set-variable** 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 **xml-to-json** policy converts the SOAP response message into a JSON object

## 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

Context variable	Name
<code>request.parameters.amount</code>	<code>amount</code>
Content type none	Definition * float
Context variable <code>request.parameters.duration</code>	Name <code>duration</code>
Content type none	Definition * integer
Context variable <code>request.parameters.rate</code>	Name <code>rate</code>
Content type none	Definition * float

+ input    + parameters for operation...    Done

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-19. Example: Define input parameters in a map policy

## 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

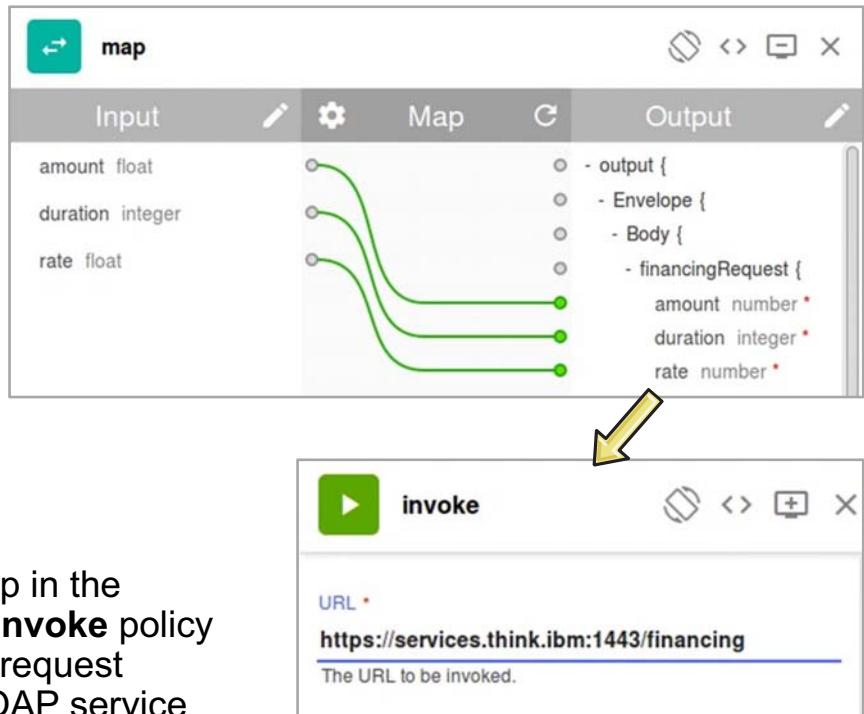
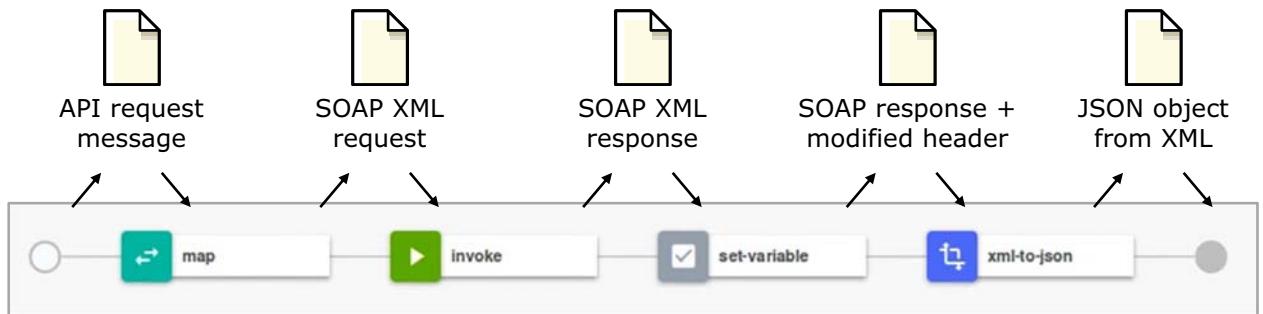


Figure 9-20. Example: Map REST parameters to a SOAP request

## 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

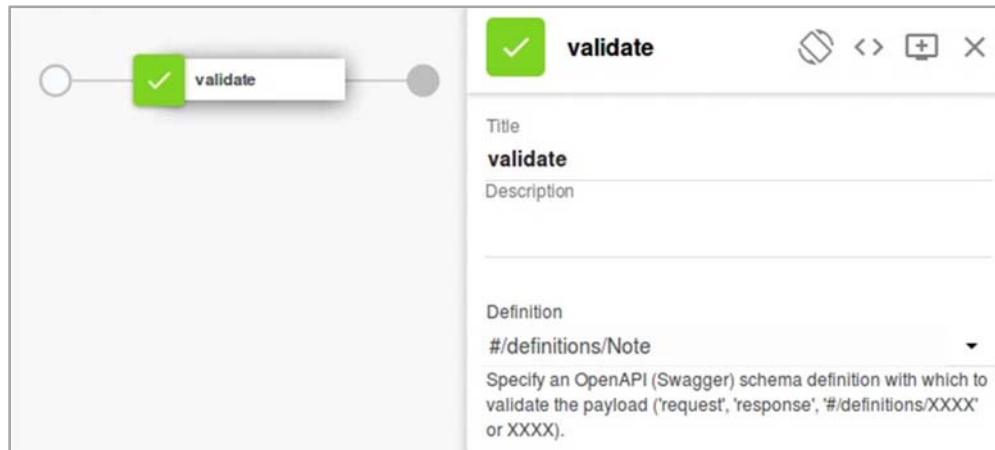


Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-21. What is the message payload?

## Example five: Validate properties in an HTTP message



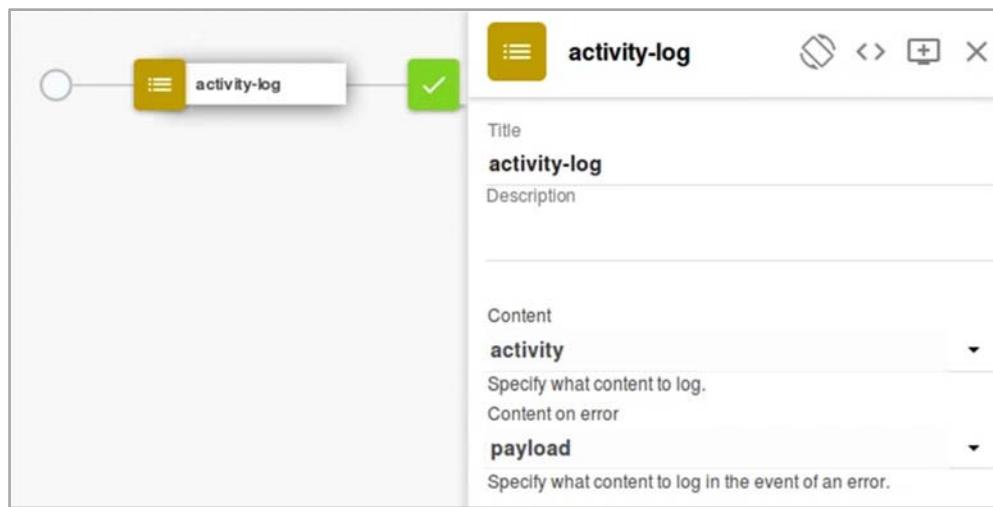
- Use the **validate** policy to check whether the message **payload** matches a defined schema type
  - To validate the original request message, add **validate** with the **request** definition at the start of the flow
  - To validate an intermediate response, add **validate** with a **custom definition**
  - To validate the API response message, add **validate** with the **response** definition after the policies that collate the final response message

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-22. Example five: Validate properties in an HTTP message

## Example six: Store message payload in API analytics



- In a **DataPower** gateway, the **activity-log** policy takes a snapshot of the assembly flow state
  - 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 in the assembly (**content on error**)

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-23. Example six: Store message payload in API analytics

The **activity-log** policy takes a snapshot of the current state in the assembly flow. 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.

A common misconception is that the activity-log policy writes API event data to the DataPower log. The policy does not write to the gateway log. You must use the API Connect analytics feature 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.

## Example seven: JSON Web Tokens

JSON Web Tokens (JWT) is an open standard (RFC 7519) that defines how to securely send data between two parties as a JSON object

- They are **compact** enough to send as a URL parameter, a POST parameter, or in the HTTP header
- The information is **digitally signed**
- The information is **self-contained**: You do not need to query a database to read the contents of a JWT token

API Connect provides two policies that work with JSON Web Tokens:

- **Generate JWT** creates a JSON Web Token
- **Validate JWT** reads a JSON Web Token, and validates the claims within the token
  - For example, you can verify the audience, the issuer, and the digital signature value

## Example: Encoded JSON Web Token (1 of 3)

```
eyJhbGciOiJIUzI1NiIsInR5cC  
I6IkpXVCJ9.eyJzdWIiOiJJQk0  
gQ2xvdWQiLCJpc3MiOiJXYXJyZ  
W4iLCJleHAiOjE0ODU4ODc3Mjh  
9.C5DYzzxVYBfrceTrmTQyNNcQ  
Zw3AtFvyD0AAKVz1XNI
```

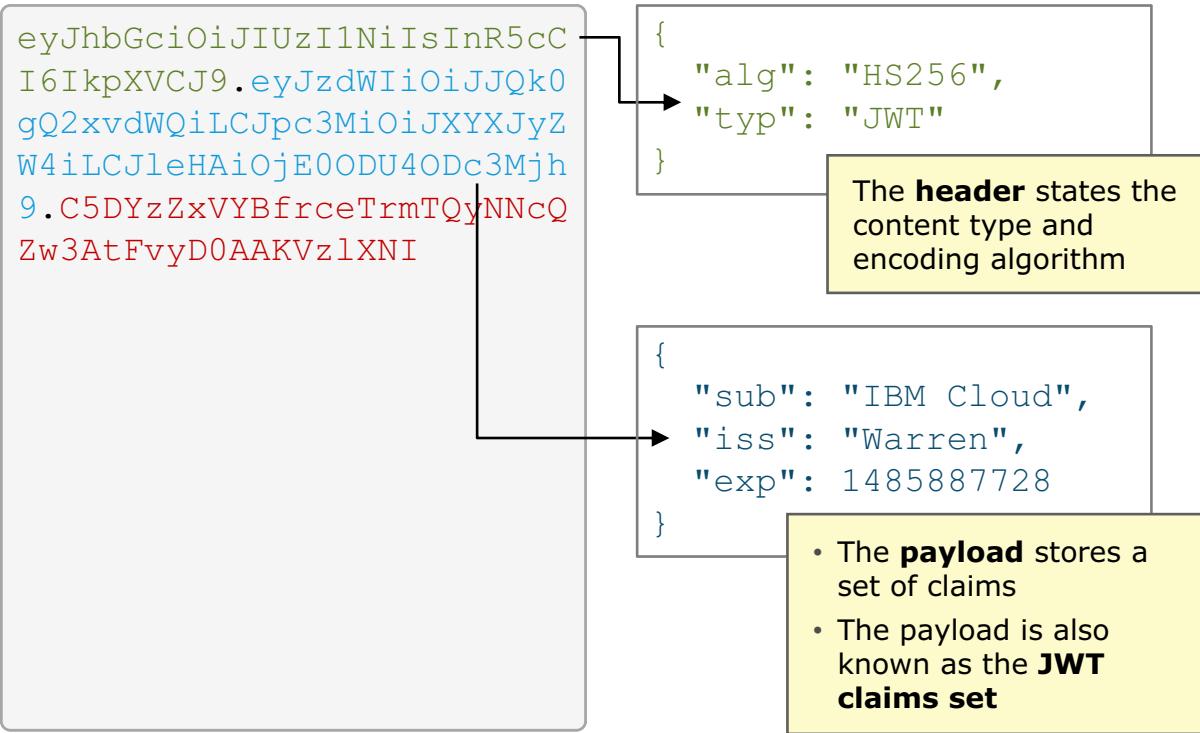
- A JSON Web Token has three sections: the **header**, the **payload**, and an **encoded signature**
- A full stop (.) separates the three parts of the token

Figure 9-25. Example: Encoded JSON Web Token (1 of 3)

The JSON Web Token consists of three parts: the header, the payload, and a digital signature for the entire token. A full stop, or period character, separates each of the three sections of the token. This encoding creates a small message that can fit into the URL parameter of an HTTP request, the POST parameter, or the request message body.

To read the information within the token, you must decode the message into a JSON object.

## Example: Encoded JSON Web Token (2 of 3)



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-26. Example: Encoded JSON Web Token (2 of 3)

The header identifies the token as a JSON Web Token. This token is encoded with the hash-based message authentication code, secure hash algorithm with 256-bit digests, or HMAC SHA-256.

The payload of the token stores a set of information that the sender claims. The information is known as the “JWT claims set”. In this example, the subject of the token is “IBM Cloud”. The issuer of the token is “Warren”. The expiry date for the token is 31 January 2017, encoded in UNIX date format.

## Example: Encoded JSON Web Token (3 of 3)

```
eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpxVCJ9.eyJzdWIiOiJJKQk0
gQ2xvdWQiLCJpc3MiOiJXYXJyZ
W4iLCJleHAiOjE0ODU4ODc3Mjh
9.C5DYzZxVYBfrceTrmTQyNNcQ
Zw3AtFvyD0AAKVz1XNI
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "IBM Cloud",
  "iss": "Warren",
  "exp": 1485887728
}
```

```
C5DYzZxVYBfrceTrmTQyNNc
QZw3AtFvyD0AAKVz1XNI
```

In API Connect, the **Validate JWT** policy verifies the token against the **encoded signature**

Figure 9-27. Example: Encoded JSON Web Token (3 of 3)

The last section of the encoded JSON Web Token is the digital signature. You check the signature value against the message to verify the integrity of the token. That is, you check whether the message is the same as the original one.

In API Connect, the **Validate JWT** policy checks the token against the encoded signature.

## Example: Generating a JSON Web Token (1 of 2)

Title: jwt-generate

Description:

JSON Web Token (JWT)

Runtime variable in which to place the generated JWT. If not set, the JWT is placed in the Authorization Header as a Bearer token.

JWT ID Claim

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

Issuer Claim: iss.claim

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

Subject Claim

Runtime variable from which the Subject (sub) claim string can be retrieved.

Audience Claim

Assembling message processing policies

You can generate a JSON Web Token with the **Generate JWT** DataPower gateway policy

- Specify a runtime variable name to save the generated token
- Otherwise, the policy adds the token in an **Authorization: Bearer** HTTP header

The **JWD ID Claim** is a unique, generated identifier stored in a claim named “**jti**”

The **Generate JWT** policy can retrieve the **issuer**, **subject**, and **audience** claims from runtime variables

© Copyright IBM Corporation 2017

Figure 9-28. Example: Generating a JSON Web Token (1 of 2)

You can generate a JSON Web Token with the Generate JWT DataPower gateway policy. Specify a runtime variable name to save the generated token. Otherwise, the policy adds the token in an **Authorization: Bearer** HTTP header. The JWD ID Claim is a unique, generated identifier stored in a claim named “**jti**”.

## Example: Generating a JSON Web Token (2 of 2)

Validity Period  
3600  
The length of time (in seconds), that is added to the current date and time, in which the JWT is considered valid.

Private Claims  
Runtime variable from which a valid set of JSON claims can be retrieved.

Sign JWK variable name  
Runtime variable containing the JWK to use to sign the JWT.

Cryptographic Algorithm\*  
Select a cryptographic algorithm.  
Sign Crypto Object  
The crypto object to use to sign the JWT.

Encryption Algorithm\*  
Select an encryption algorithm.  
Encrypt JWK variable name  
Runtime variable containing the JWK to use to encrypt the JWT.

Key Encryption Algorithm\*  
Select a key encryption algorithm.  
Encrypt Crypto Object

Assembling message processing policies

The **Generate JWT** policy takes the validity period to calculate the **expiry date** for the token

Use a runtime variable to save the cryptographic key that the policy uses to generate the **token digital signature**

Specify the **cryptographic algorithm** that the policy uses to generate the signature

You can also specify RSA or SHA keys that are stored in the DataPower gateway in the **Sign Crypto Object** field

With the **encryption algorithm** and **JSON Web Token key (JWK)** fields, the policy encrypts the token

© Copyright IBM Corporation 2017

Figure 9-29. Example: Generating a JSON Web Token (2 of 2)

The Generate JWT policy takes the validity period to calculate the expiry date for the token. Use a runtime variable to save the cryptographic key that the policy uses to generate the **token digital signature**. Specify the **cryptographic algorithm** that the policy uses to generate the signature.

Specify the cryptographic algorithm that the policy uses to generate the signature.

With the encryption algorithm and JSON Web Token key (JWK) fields, the policy encrypts the token.

## User-defined policies

- Beyond the list of built-in message processing policies, you can create your own actions in a **user-defined policy**
  - In a **DataPower** gateway, a user-defined policy is a **processing rule**
  - In a **Micro** gateway, a user-defined policy is a **Node.js** module
- The main reason to create user-defined policies is to **create or access more capability** beyond the built-in set of policies
  - Define your own content-based routing logic in a single policy
  - Enforce authorization constraints beyond the built-in security policies
  - Access extra DataPower features in your API Connect assembly

Figure 9-30. User-defined policies

Beyond the list of built-in message processing policies, you can create your own actions in a user-defined policy. You write custom logic that is specific to the API gateway type. In a DataPower gateway, a user-defined policy is a processing rule. In a Micro gateway, a user-defined policy is a Node.js module.

The main reason to create user-defined policies is to **create or access extra capability** beyond the built-in set of policies. Define your own content-based routing logic in a single policy. Enforce authorization constraints beyond the built-in security policies. Access more DataPower features in your API Connect assembly.

## Example: writeToDatapowerLog policy

- The **writeToDatapowerLog** policy reads a string from an API Connect assembly flow and writes it into the DataPower system log
- Unlike the **activity-log** policy, you can write your own messages, log environment variables, or save troubleshooting data to a log

The screenshot shows the IBM API Connect Policy Editor interface. On the left, the policy configuration pane displays:

- logLevel**: error
- message**: `$(api.name) appears to have malfunctioned.`
- version**: 1.0.0

A yellow arrow points from the "message" field to the log entry in the pane on the right. The right pane shows the log entries:

msgid	message	Show last
0x8580005c	mpgw (webapi): The API: "writetodatapowerlog-api" has logged the following message: writetodatapowerlog-api appears to have malfunctioned.	50 100 all
0x8580005c	mpgw (webapi): The API: "writetodatapowerlog-api" has logged the following message: Initial Critical Message	

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-31. Example: writeToDatapowerLog policy

For more information, see “Increase logging with a custom policy for IBM DataPower in the API Connect assembly”: <https://www.ibm.com/developerworks/library/mw-1610-phillips-trs/index.html>

## Example: `writeToDatapowerLog` custom policy

```
function writeToDatapowerLog() {
    var apic = require('local://isp/policy/apim.custom.js');
    var props = apic.getPolicyProperty(),
        apiName = apic.getContext('api.name'),
        msg = props.message;

    if (props === undefined) {
        console.error('Cannot retrieve policy properties.');
        return;
    }
    if (apiName === undefined || apiName === '') {
        console.error('Cannot determine the name of the API.');
    }

    console.log('API \"' + apiName + '\", message: ' + msg
    return;
}

writeToDatapowerLog();
```

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-32. Example: `writeToDatapowerLog` custom policy

This slide displays the Gateway script code that implements the **`writeToDatapowerLog`** policy.

The first line in the `writeToDatapowerLog` function imports the **`apim.custom.js`** JavaScript library and assigns it to the **`apic`** object. The **`apic`** policy retrieves runtime information from API Connect. For example, the `apic.getPolicyProperty` function retrieves the properties for the **`writeToDatapowerLog`** policy. The `apic.getPolicyProperty.message` property represents the custom log message that the API developer defined in the custom policy.

The second-to-last line writes the name of the API that was called at the gateway, and the custom message.

For more information, review the “Implementing your policy” article in the IBM Knowledge Center: [https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.policy.doc/tapim\\_policy\\_implement.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.policy.doc/tapim_policy_implement.html)

## Assembly palette: Logic constructs

Type	Description	How to use
if	Apply a section of the assembly when a condition is fulfilled	<b>Condition</b> property set by JavaScript or Gateway script
operation-switch	Apply a section of the assembly based on the API operation that is being called	<b>Case</b> statements are API operations: a combination of <b>verb</b> and <b>path</b> properties
switch	Apply a section of the assembly based on an arbitrary condition	<b>Case</b> statements are JavaScript or Gateway Script
throw	Throw an error to return an error object	Set the <b>name</b> and <b>message</b> fields with the error details

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-33. Assembly palette: Logic constructs

The **logic constructs** direct the flow of the message processing policies.

The **if** construct applies a section of the assembly when a condition is fulfilled. You set the **condition** property as JavaScript or Gateway script.

The **operation-switch** construct applies a section of the assembly based on the API operation that is being called. You define **case** statements: a combination of **verb** and **path** properties that trigger a case.

The **switch** construct applies a section of the assembly based on a logic statement. You define the **case** statement in JavaScript or Gateway script.

The **throw** construct throws an error to return an error object. You set the **name** and **message** fields with the error details.

## Assembly palette: API policies for invocation and code

Type	Description	How to use
set-variable	Set a runtime variable to a string value	Set, add, or clear a runtime variable
invoke	Call a remote HTTP endpoint	Set <b>target-url</b> to the remote host name and base path
proxy DataPower	Forward requests to another API endpoint	<ul style="list-style-type: none"> <li>Set <b>target-url</b> to the remote host name and base path</li> <li>Only one proxy per assembly</li> </ul>
activity-log DataPower	Store API activity in API Connect analytics	Set <b>content</b> property to <b>activity</b> , <b>header</b> , <b>payload</b> , or <b>none</b>
gatewayscript DataPower	Run Gateway Script code	Add valid Gateway Script code into the policy
validate	Validate the current message payload against a defined schema type	Specify the schema of the <b>request</b> , <b>response</b> , or custom <b>definition</b> to compare against

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-34. Assembly palette: API policies for invocation and code

The **invocation** and **code** policies make external API requests, or run arbitrary logic that is written in JavaScript.

The **set-variable** policy sets a runtime variable with a string value. Use the **set**, **add**, or **clear** actions on a runtime variable.

The **invoke** policy calls a remote HTTP endpoint. Set the **target-url** property to the remote host name and base path.

The **proxy** policy forwards requests to another API endpoint. Set the **target-url** property to the remote host name and base path. You can add one proxy policy per assembly only.

The **activity-log** policy stores API invocation details in API Connect analytics. Set the **content** property to **activity header**, **payload**, or **none**.

The **gateway script** and **JavaScript** policies run arbitrary code at the gateway.

Only the DataPower API Gateway supports the **proxy**, **activity-log**, and **gateway script** policies.

Only the Micro gateway supports the **JavaScript** policy.

The **validate** policy checks the current message payload structure against a defined schema type. You specify the schema of the request, response, or custom definition to compare against.

## Assembly palette: API policies for message mapping

Type	Description	How to use
json-to-xml DataPower	Convert the content payload from JSON to XML format	Policy reads and converts the message body automatically
map DataPower	Copy, concatenate, and transform properties from one message to another	Use the <b>map</b> editor in the Assembly view to map fields from the input message to an output message
xml-to-json DataPower	Convert the content payload from XML to JSON format	Policy reads and converts the message body automatically
xslt DataPower	Apply an XML stylesheet transform to an XML content payload	Specify an existing or inline XSLT stylesheet to apply to the content payload
Redaction DataPower	Completely remove or block out certain sections of the message body or activity logs	<ul style="list-style-type: none"> <li>Add a <b>remove</b> or <b>redact</b> action</li> <li>Specify an XPath expression in the <b>path</b> statement</li> </ul>

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-35. Assembly palette: API policies for message mapping

The **message mapping** policies transform the fields or data types of the API request or response message.

The **json-to-xml** policy converts the content payload from JSON to XML format. The policy reads and converts the message body automatically.

The **map** policy copies, concatenates, and transforms properties from one message to another. Use the **map** editor in the Assembly view to map fields from the input message to an output message.

The **xml-to-json** policy converts the content payload from XML to JSON format. The policy reads and converts the message body automatically.

The **xslt** policy applies an XML stylesheet transform to an XML content payload. You specify an existing or inline XSLT stylesheet to apply to the content payload.

The **json-to-xml**, **xml-to-json**, **map**, and **xslt** policies appear in the DataPower gateway only. The **validate** policy is available in both the Micro gateway and DataPower gateway.

The **Redaction** policy completely removes or blocks out certain sections of the message body or activity logs. You add a **remove** or **redact** action and specify an XPath expression in the **path** statement.

## Assembly palette: API policies for security

Type	Description	How to use
Generate JWT DataPower	Generate a JSON Web Token	Refer to documentation
Validate JWT DataPower	Validate a JSON Web Token	Refer to documentation
Generate LTPA token DataPower	Generate a lightweight third-party authentication (LTPA) token	Set the LTPA key from which to generate the token
Validate Username token DataPower	Validate a WS-Security UsernameToken in a SOAP message header	Specify the LDAP registry or authentication URL to verify the token value

Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-36. Assembly palette: API policies for security

The **security** policies verify or generate security tokens, and secure message contents.

The **Generate JWT** policy generates a JSON Web Token. Refer to the example in this presentation for the generate policy properties.

The **Validate JWT** policy checks whether a JSON Web Token is valid. For more information, see the IBM Knowledge Center.

The **Generate LTPA token** policy generates a Lightweight Third Party Authentication (LTPA) token. Set the LTPA key from which to generate the token.

The **Validate Username token** policy validates a WS-Security UsernameToken header in a SOAP message. You specify the LDAP registry or authentication URL to verify the token value.

Only the DataPower gateway supports the policies that are listed on this slide.

## 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
- List the policies that each API gateway type supports
- Explain how to generate JSON Web Tokens with policies
- Explain the concept and use case for custom user-created policies

## 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 response body
  - D. The payload is a buffer that the policy assembly uses to process or construct an API response message
3. Which policy is available only on DataPower gateways?
  - A. Validate
  - B. Map
  - C. Invoke
  - D. Set Variable



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-38. 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 response body
  - 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 available only on DataPower gateways?
  - A. Validate
  - B. Map
  - C. Invoke
  - D. Set Variable  
The answer is B.



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-39. Review answers

## Exercise: Assembling message processing policies

Assembling message processing policies

© Copyright IBM Corporation 2017

*Figure 9-40. Exercise: Assembling message processing policies*

## Exercise objectives

- 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
- Map data from multiple API calls into an aggregate response
- Define and call a gateway script with an API assembly
- Test DataPower gateway policies in the API Manager explore view



Assembling message processing policies

© Copyright IBM Corporation 2017

Figure 9-41. 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 2017

Figure 10-1. Unit objectives

## 10.1. API security concepts

## API security concepts

Declaring client authorization requirements

© Copyright IBM Corporation 2017

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

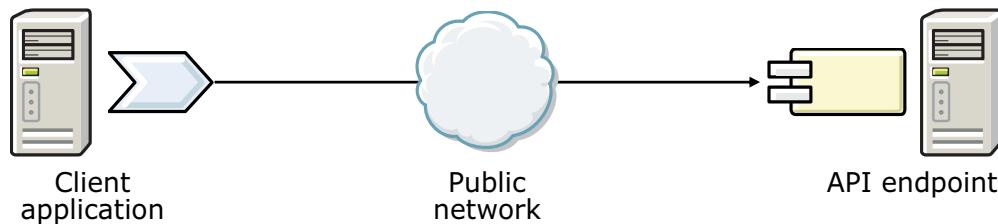
[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2017

*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 permit
- 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 2017

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 2017

Figure 10-5. How do you secure your APIs in API Connect?

The security definitions that you create in an API definition configure client authentication and authorization schemes.

## What types of security definitions can you define?

Definition type	Description
API key	The <b>API key</b> scheme authenticates the API caller from the <b>client ID</b> and <b>client secret</b> credentials
Basic	The <b>HTTP basic authentication</b> scheme enforces authentication and authorization at the HTTP message protocol layer
OAuth 2.0	The <b>OAuth 2.0</b> 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 2017

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 2017

*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 2017

Figure 10-8. Topics

## API key: A unique client application identifier

The screenshot shows the left sidebar of the API Connect interface with various options like Info, Schemes, Host, etc., and the 'Security Definitions' section highlighted with an orange box. The main panel displays a table for a security definition named 'clientID (API Key)'. The table rows include Name (clientID), Parameter name (X-IBM-Client-Id), Located In (Header), and Description.

Security Definitions	
clientID (API Key)	
Name	clientID
Parameter name	X-IBM-Client-Id
Located In	Header
Description	

- 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**
  - You can also define a security definition and requirement for a **client secret**

Declaring client authorization requirements

© Copyright IBM Corporation 2017

Figure 10-9. API key: A unique client application identifier

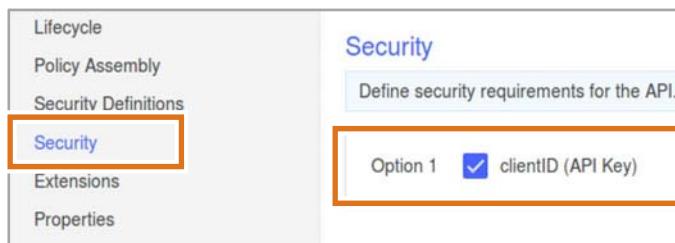
The API key uniquely identifies the client application. If you enable **clientID** as a security requirement in your API, the client must provide its client ID on every API operation call. In addition to establishing the client identity, API Connect uses the client ID value for analytics and to enforce 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.

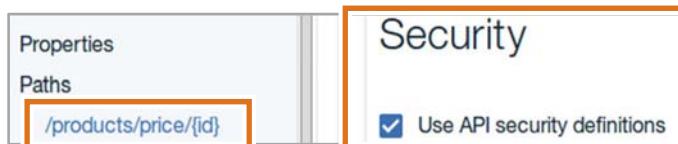
To apply an API key security requirement to your API, select the **security definitions** section in your API definition. Create an **API key** security definition for either a client ID or a client secret. When you create an API definition, the API Designer adds a security definition for a **client ID** for you.

## Enforcing security definitions

- You must **enable** the security definition to enforce the security scheme in your API
  - Select the security definition option in the **security** section of your API definition



- 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



Declaring client authorization requirements

© Copyright IBM Corporation 2017

Figure 10-10. Enforcing security definitions

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. 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

## 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 10-12. 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 2017

*Figure 10-13. 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

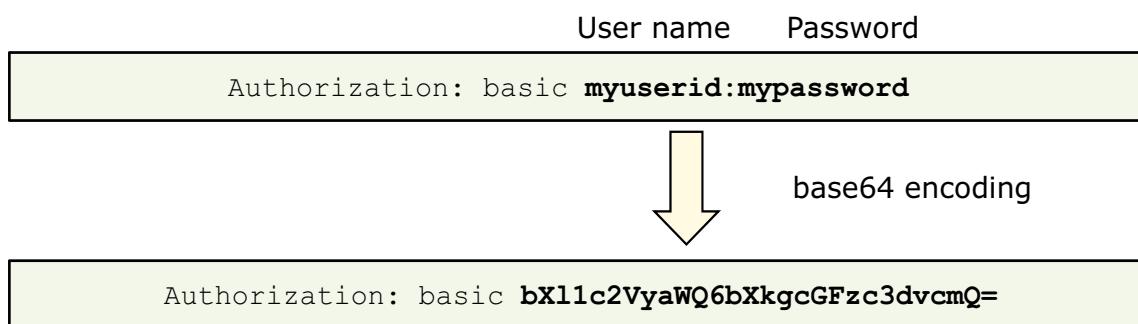
Declaring client authorization requirements

© Copyright IBM Corporation 2017

Figure 10-14. 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 2017

Figure 10-15. 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



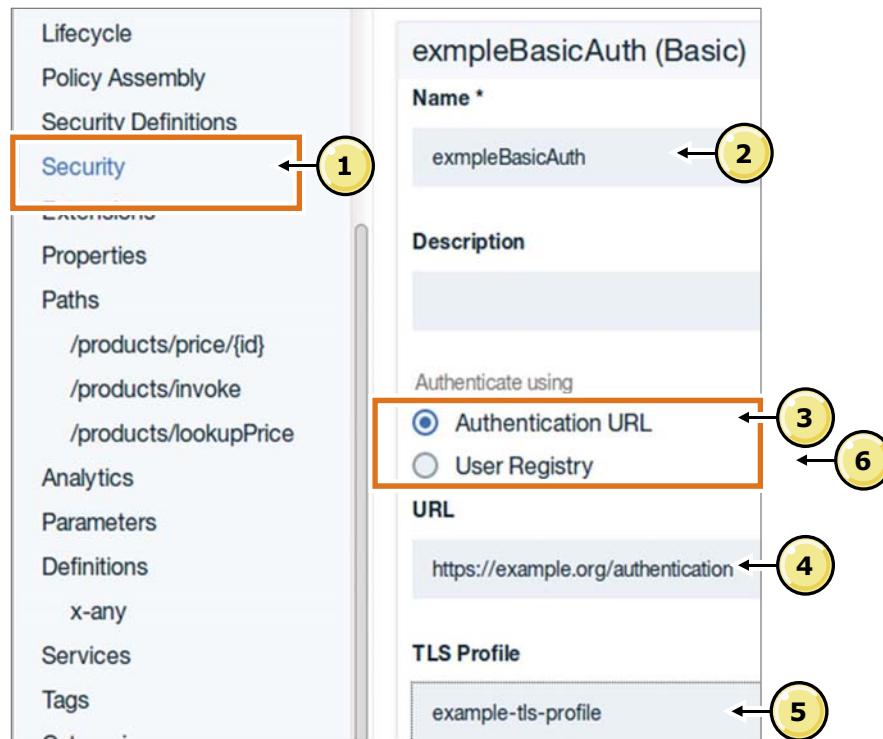
Declaring client authorization requirements

© Copyright IBM Corporation 2017

Figure 10-16. 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.

## Example: Basic authentication security definition



Declaring client authorization requirements

© Copyright IBM Corporation 2017

Figure 10-17. Example: Basic authentication security definition

To set up a Basic authentication scheme for your API, create a **Basic** security definition.

1. In the API Designer web application, select the **Security definitions** section of your API definition. Create a **Basic** security definition.
2. Enter a unique name and description for your Basic authentication security definition.
3. Choose one of two options to authenticate client applications: an **Authentication URL**, or an **LDAP User Registry**. In this example, you select the **Authentication URL** option.
4. Enter 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.
5. 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.
6. 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.

## Message confidentiality with Transport Layer Security

- The Transport Layer Security (TLS) protocol encrypts data that is sent from an HTTP client to a web server
  - The TLS protocol is a newer update to the secure sockets layer (SSL) specification
  - When you enter `https` as the protocol in the web browser address bar, the browser sets up a TLS connection
- The client and the web server exchange a shared key to encrypt all HTTP messages
  - For an added level of security, the server can authenticate the client's identity by inspecting its digital certificate
- TLS provides a secure, point-to-point connection irrespective of the web service style
  - You can send both REST and SOAP web services messages over a secure HTTP connection

[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2017

*Figure 10-18. Message confidentiality with Transport Layer Security*

Although the HTTP basic authentication protocol does not require an encrypted connection, it is advised as it sends sensitive data over the network.

Typically, websites do not verify the identity of the client. When you log on to your online banking account, the bank's website does not verify the identity of your computer with a digital certificate. If you provide a correct user name and password, the website assumes that you are the account holder. The bank uses a secure connection to communicate with your client.

Since Transport Level Security operates at the transport level, you can use it with any web service style: either REST or SOAP.

The Transport Layer Security protocol version 1.0 is an upgrade to the Secure Sockets Layer (SSL) version 3.0 protocol. All web browsers support at least TLS 1.0. You are advised to use TLS 1.0 on your website as the minimum version for securing HTTP connections.

The Cloud Manager and API Connect both support and use SSL certificates but do not themselves produce strong encryption keys or manage your encryption keys. Encryption keys should be generated and managed according to your own company procedures.

## 10.4. Introduction to OAuth 2.0

## Introduction to OAuth 2.0

Declaring client authorization requirements

© Copyright IBM Corporation 2017

*Figure 10-19. 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 2017

Figure 10-20. 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 2017

Figure 10-21. 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

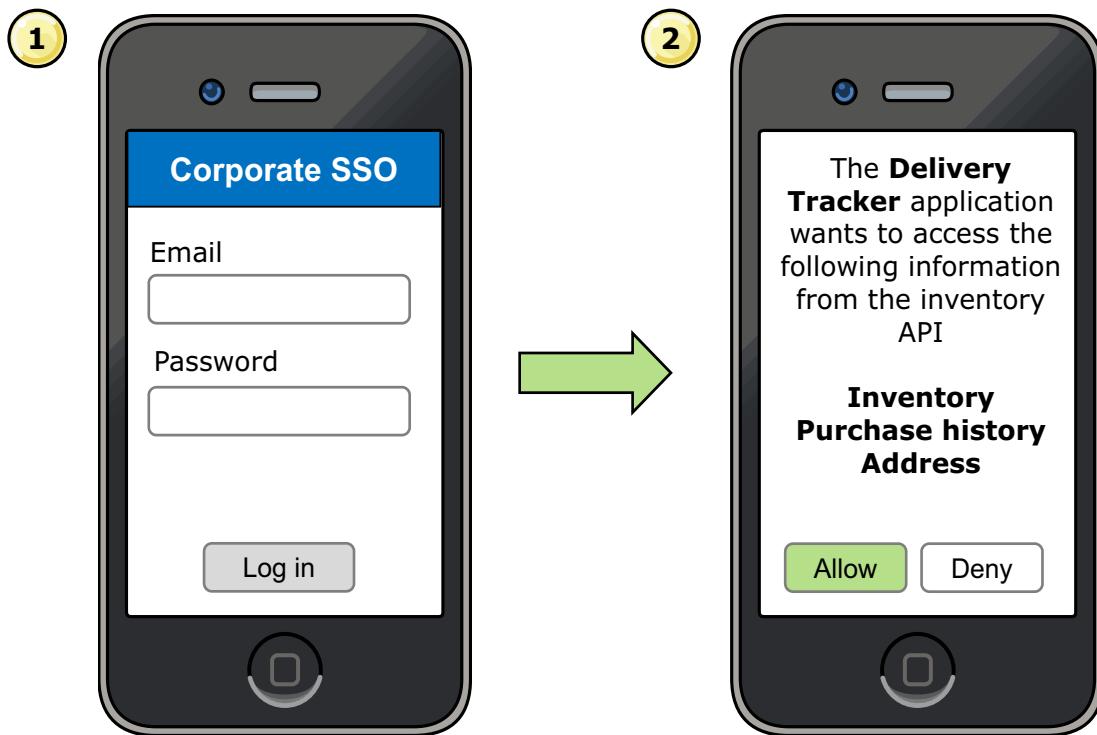
OAuth separates the role of the client from the role of the resource owner.

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 2017

Figure 10-22. 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



- Delivery tracker, a third-party **client application**, wants to access Alice's inventory records from the API



- The inventory API provides online access to inventory records
- This service acts as the **resource server**
- It manages access to Alice's records

- An **authorization server** 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

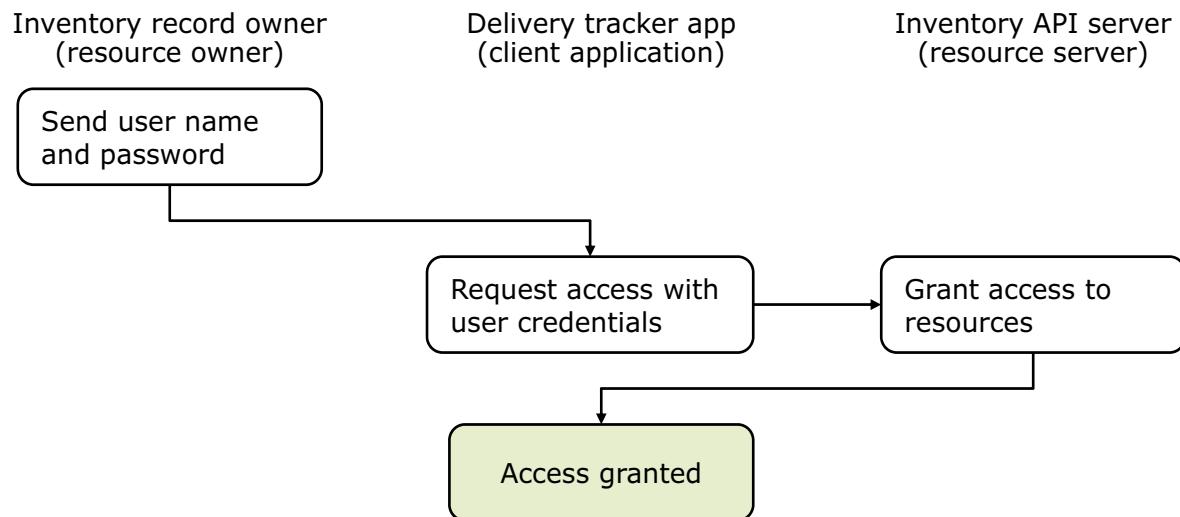
[Declaring client authorization requirements](#)

© Copyright IBM Corporation 2017

Figure 10-23. 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.

## Before OAuth: Sharing user passwords



- **Issues with sharing passwords:**
  - The service cannot distinguish between the user (resource owner) and the third-party application
  - No method to revoke access for just the third-party application

[Declaring client authorization requirements](#)

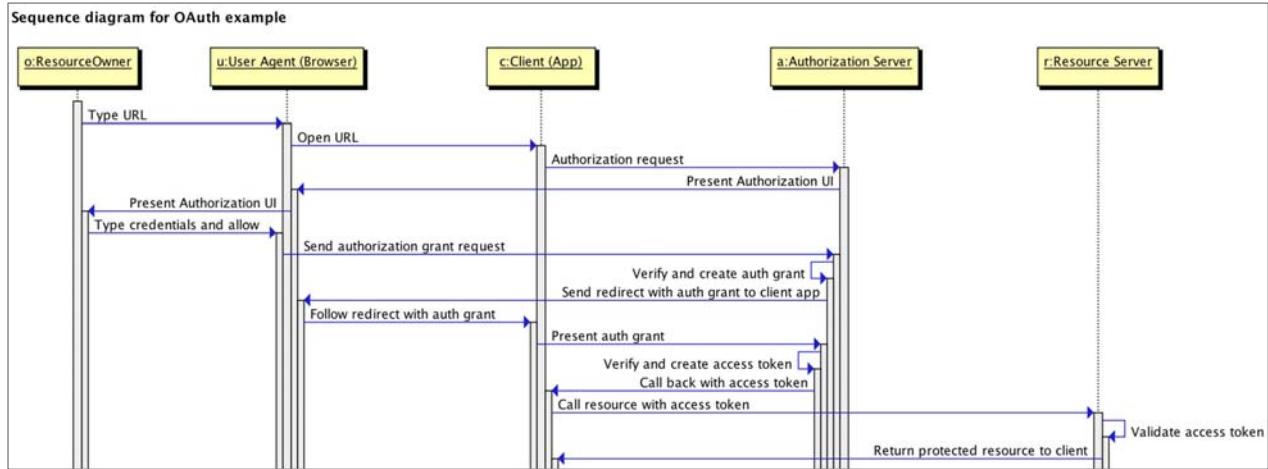
© Copyright IBM Corporation 2017

Figure 10-24. Before OAuth: Sharing user passwords

Without OAuth, the user must give the user name and password to the third-party application. In turn, the third-party application sends these credentials while posing as the user. For the sake of convenience, the application saves a copy of the user name and password.

This scenario has several issues. First, the service cannot distinguish between the owner of the resource and the third-party application. To the service, it is the same user that is accessing the application. This practice is not safe; the user does not know what the application reads or modifies on the service. Second, you have no simple way to revoke access for one particular third-party application. The user must reset the password, which breaks access from all third-party applications.

## OAuth flow sequence



Declaring client authorization requirements

© Copyright IBM Corporation 2017

Figure 10-25. OAuth flow sequence

With OAuth, the client application is not given the user credentials directly.

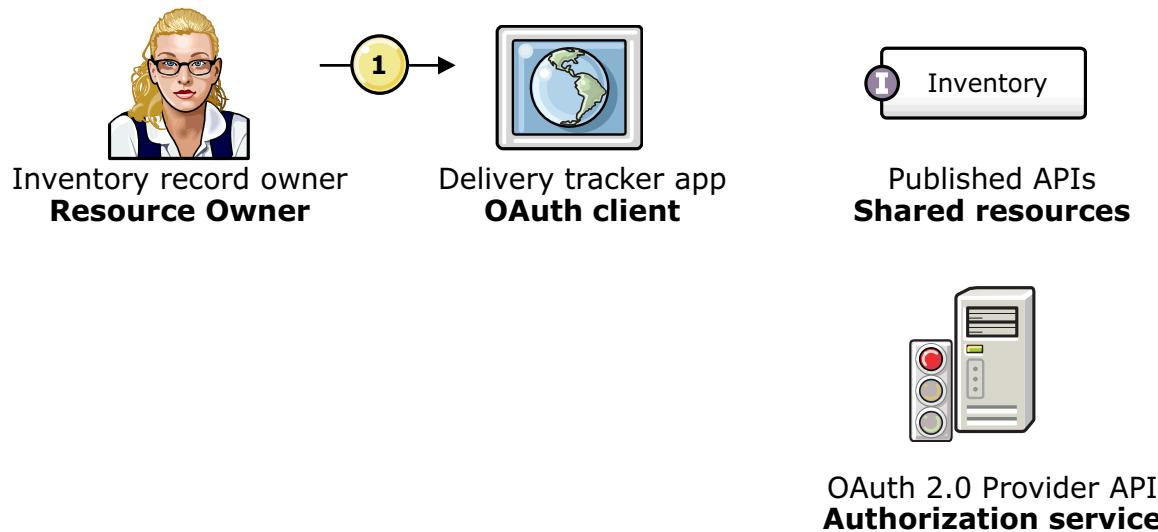
The diagram shows the interaction between the resource owner and the browser and the client application. The flow of messages between the client application and the servers is also shown.

The next sequence of slides provides details of the steps with an example.

This page shows a call sequence diagram for OAuth. The interactions occur between the client application and its hosting browser, and the back-end authorization and resource servers. The resource owner chooses to allow or deny access to the resource. In some cases, the authorization server and resource server belong to the same back-end service.

## 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

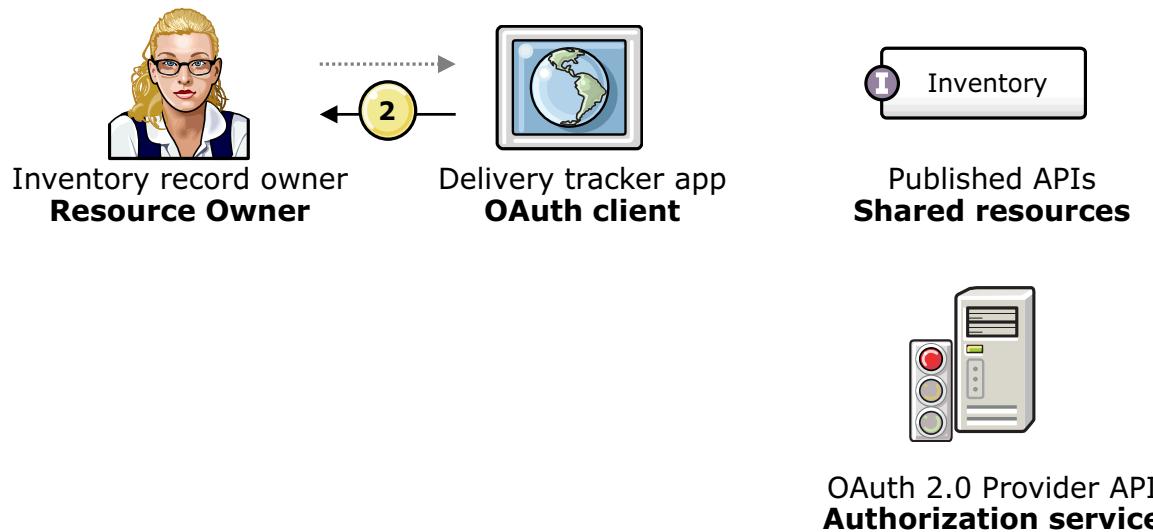
© Copyright IBM Corporation 2017

Figure 10-26. 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 third-party OAuth client application. Alice starts the process when she selects the “look up your deliveries” option in the third-party application.

## OAuth Step 2: OAuth client redirection to owner

- Step 2: The OAuth client sends the resource owner a redirection to the authorization server



[Declaring client authorization requirements](#)

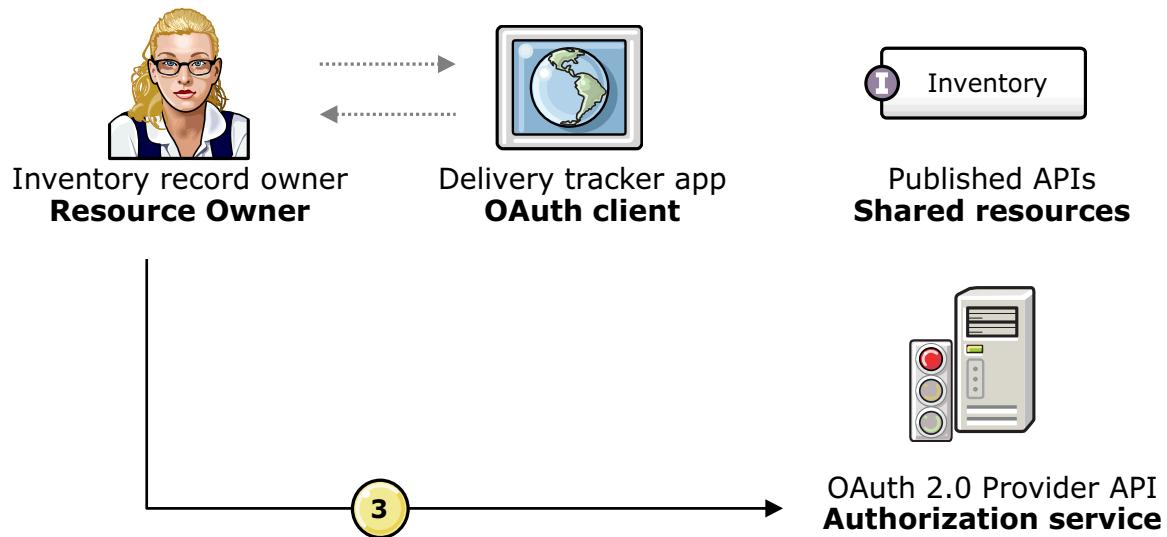
© Copyright IBM Corporation 2017

Figure 10-27. OAuth Step 2: OAuth client redirection to owner

In the second step, the third-party 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 third-party client application redirects Alice's request to an authorization server.

## OAuth Step 3: Authenticate owner with authorization server

- Step 3: The resource owner authenticates against the authorization server



Declaring client authorization requirements

© Copyright IBM Corporation 2017

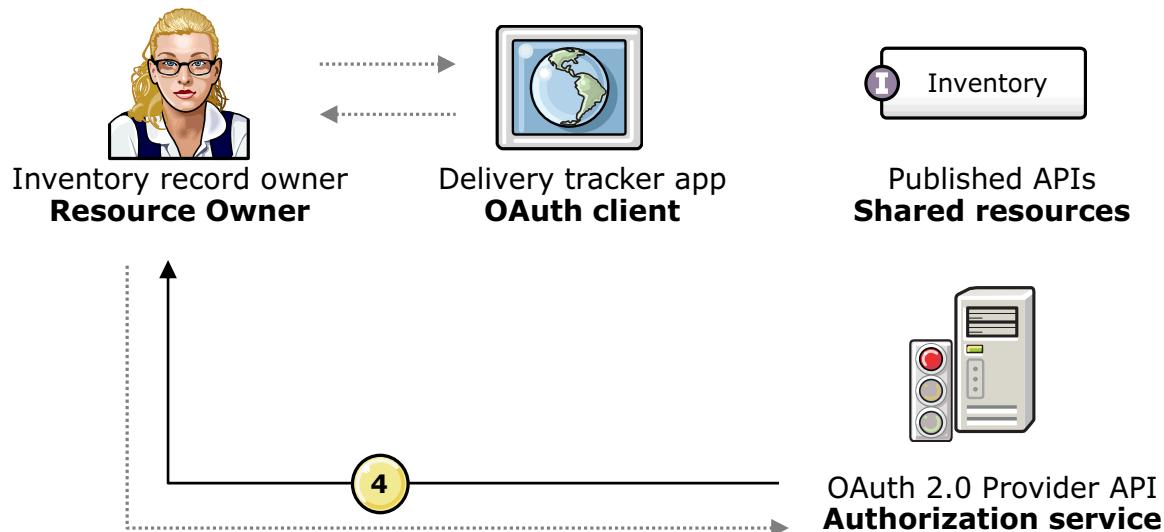
Figure 10-28. OAuth Step 3: Authenticate owner with authorization server

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 server returns a web form to the resource owner to grant access



Declaring client authorization requirements

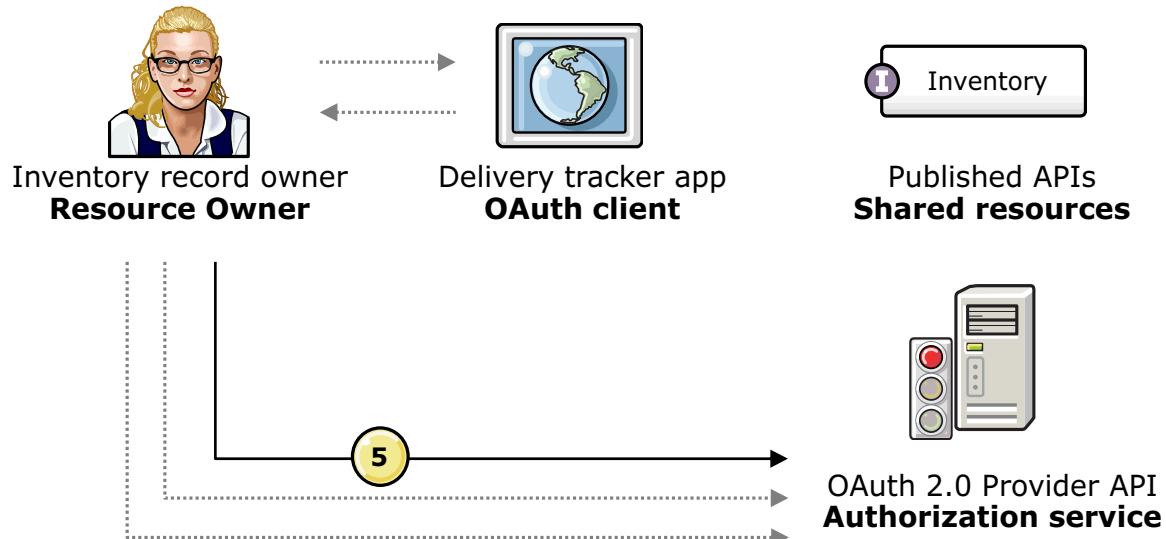
© Copyright IBM Corporation 2017

Figure 10-29. OAuth Step 4: Ask resource owner to grant access to resources

The authorization server 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

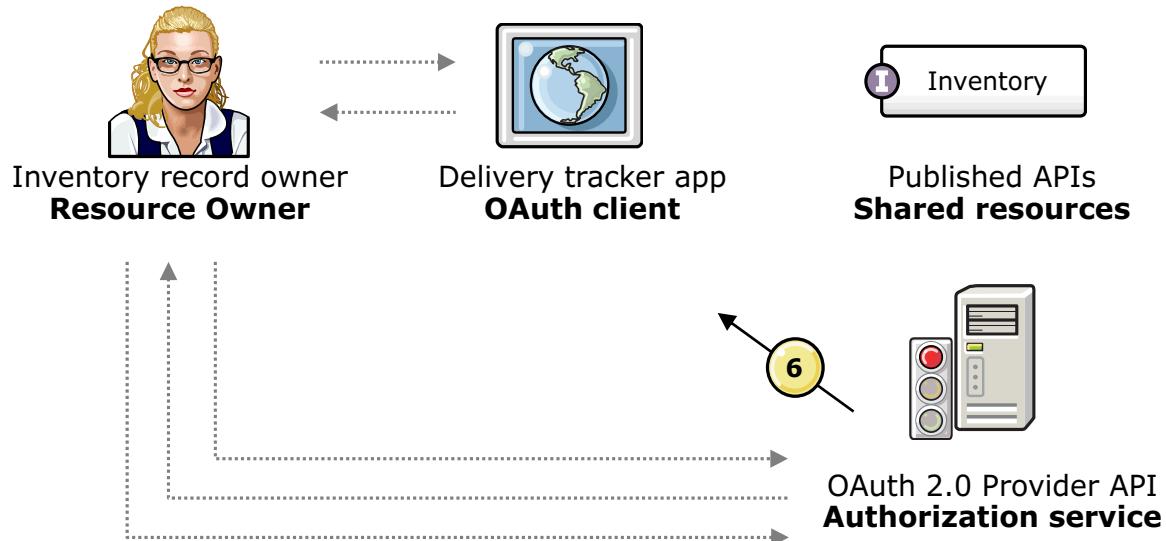
© Copyright IBM Corporation 2017

Figure 10-30. OAuth Step 5: Resource owner grants client access to resources

The resource owner, Alice, submits the web form to permit or deny access to her resources.

## OAuth Step 6: Authorization server sends authorization grant code to client

- Step 6: If the resource owner allows access, the authorization server sends the OAuth client a redirection with the authorization grant code



Declaring client authorization requirements

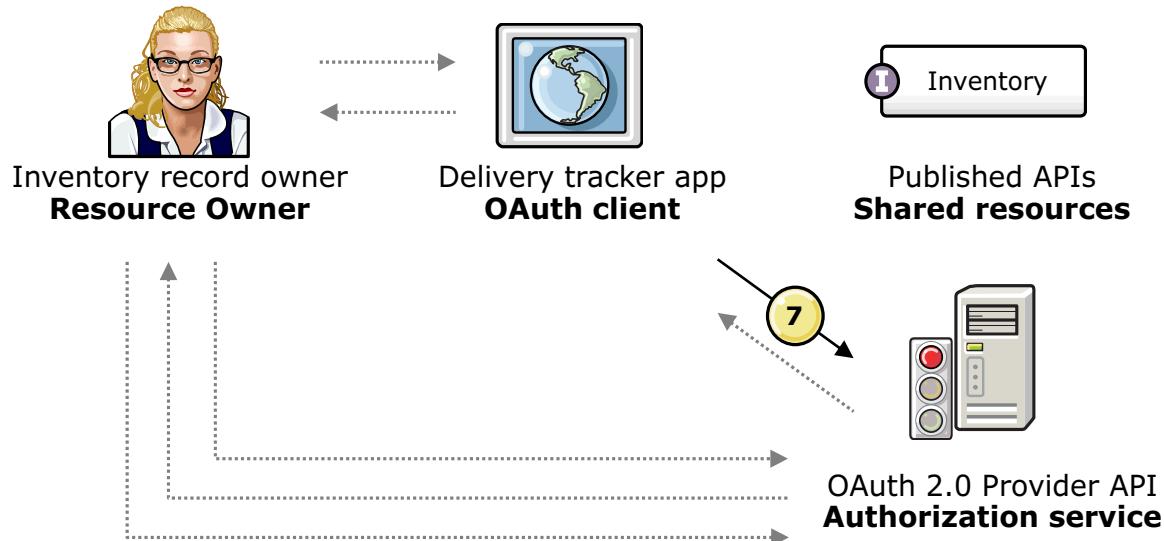
© Copyright IBM Corporation 2017

Figure 10-31. OAuth Step 6: Authorization server sends authorization grant code to client

The authorization server never transmits the resource owner's user name and password to the OAuth client. Instead, the server 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 server

- Step 7: To access the resource, the OAuth client sends the authorization grant code and other information to the authorization server



Declaring client authorization requirements

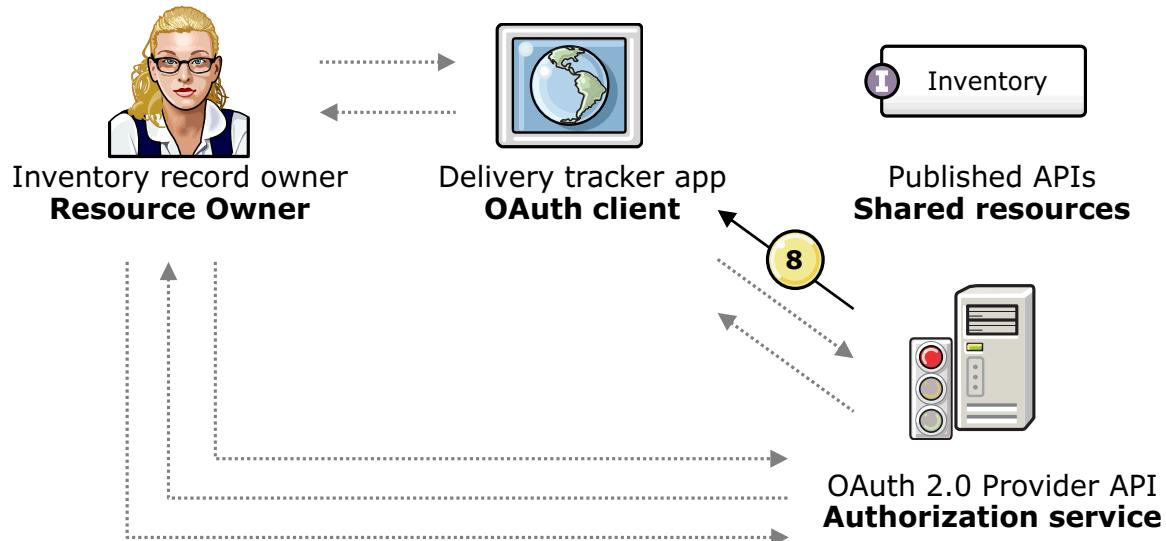
© Copyright IBM Corporation 2017

Figure 10-32. OAuth Step 7: Client requests access token from authorization server

The OAuth client sends three pieces of information to the authorization server: an authorization grant code, the client ID, and the client secret or client certificate. If the OAuth client is a public client, then it does not send the client secret or certificate.

## OAuth Step 8: Authorization server sends authorization token to client

- Step 8: If the authorization server verifies the grant authorization information, it returns an access token to the OAuth client



Declaring client authorization requirements

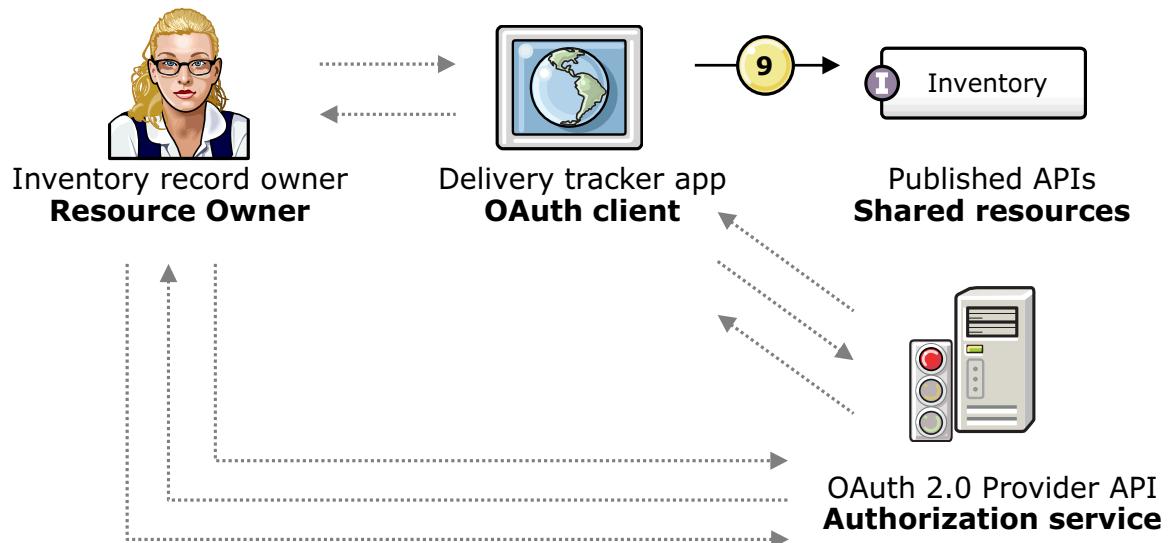
© Copyright IBM Corporation 2017

Figure 10-33. OAuth Step 8: Authorization server sends authorization token to client

Optionally, the authorization server can also return a refresh token. After the current access token expires, the OAuth client sends the refresh token to the authorization server to request another access token.

## OAuth Step 9: OAuth client sends access token to resource server

- Step 9: The OAuth client sends the access token to the resource server



Declaring client authorization requirements

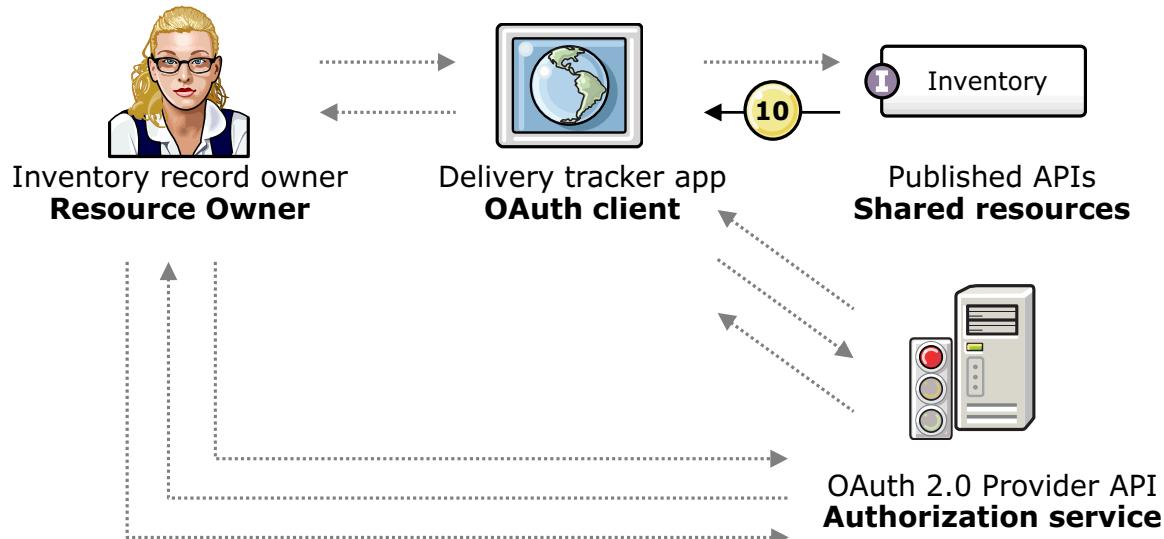
© Copyright IBM Corporation 2017

Figure 10-34. OAuth Step 9: OAuth client sends access token to resource server

It is possible that the authorization server and the resource server are 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 2017

Figure 10-35. OAuth Step 10: Resource server grants access to OAuth client

Optionally, the authorization server can also return a refresh token. After the current access token expires, the OAuth client sends the refresh token to the authorization server to request another access token.

## 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 2017

Figure 10-36. 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 2017

Figure 10-37. Review questions

Write your answers here:

1.

2.

## 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

01:00

## 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 an OAuth 2.0 Provider in an API definition from the API Designer web application.

## How you will check your progress

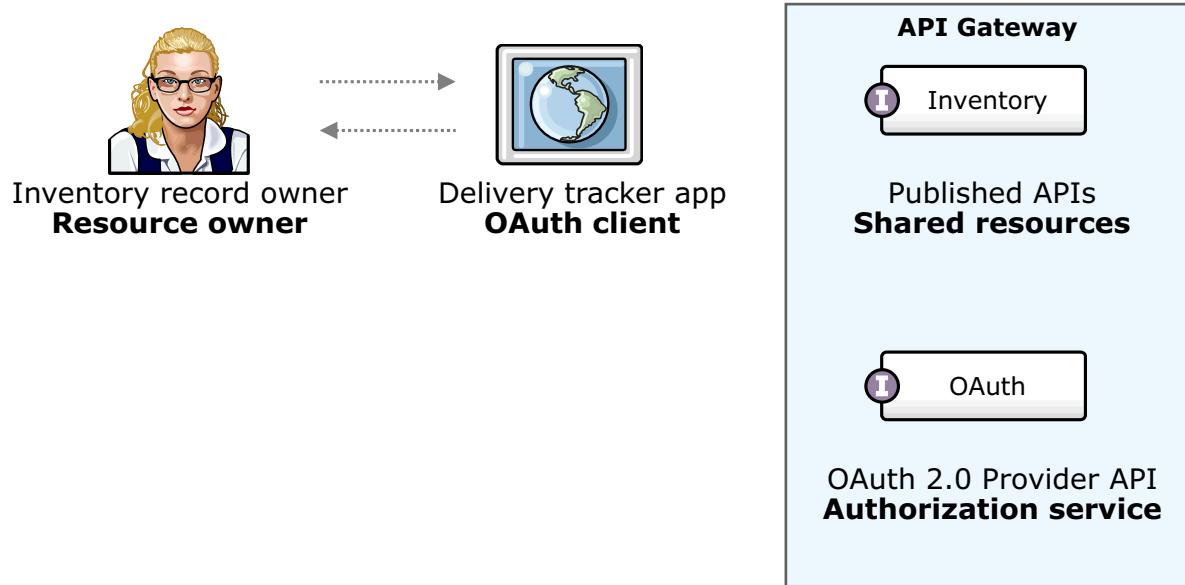
- Review questions
- Lab exercise

## Unit objectives

- Explain the concept of an OAuth provider
- Identify the role of the authorization server in the OAuth flow
- Identify the OAuth 2.0 roles, client types, and schemes
- Explain the difference between public and confidential schemes
- Explain the OAuth flow and grant types
- Explain the concept of access tokens
- Explain the concept of the token revocation service
- Explain the purpose of the token introspection endpoint
- Explain the concept of OAuth metadata
- Explain the relationship between the OAuth provider and the API security definition

## Role of IBM API Connect in the OAuth flow

- 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 2017

Figure 11-2. Role of IBM API Connect in the OAuth flow

In IBM API Connect, the OAuth 2.0 Provider API implements the **authorization** and **token** services in an OAuth flow. If you already have an OAuth 2.0 Provider, you can configure the OAuth 2.0 security definition to call your existing authorization service instead.

## 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. 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 with an API definition
- The API gateway implements the OAuth 2.0 Provider API operations

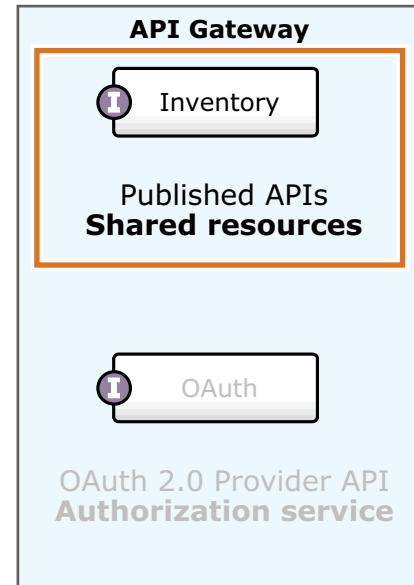
### 2. For each API that you want to secure, declare an **OAuth 2.0 security definition**

- 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

Figure 11-3. What are the steps to secure an API with OAuth 2.0?

## 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 network address of the **OAuth 2.0 Provider**
  - Which API **operations to secure**
  - Which OAuth 2.0 **message flow** to use
  - **How to authenticate** the identity of the calling application
  - **How to authorize access** to authenticated users
  - **Token lifespan and revocation settings**
  - Extra **metadata** that the client encodes in an OAuth token



[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

Figure 11-4. What is an OAuth 2.0 security definition?

### What is an OAuth 2.0 security definition?

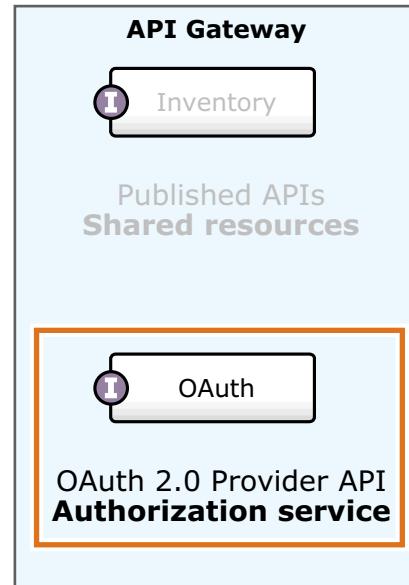
The **OAuth 2.0 security definition** specifies the security settings in the API that you want to secure.

The following list describes the OAuth 2.0 configuration settings that you can customize in the security definition:

- The network address of the **OAuth 2.0 Provider**
- Which API **operations to secure**
- Which OAuth 2.0 **message flow** to use
- **How to authenticate** the identity of the calling application
- **How to authorize access** to authenticated users
- **Token lifespan and revocation settings**
- Extra **metadata** that the client encodes in an OAuth token

## What is an OAuth 2.0 Provider?

- The OAuth 2.0 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

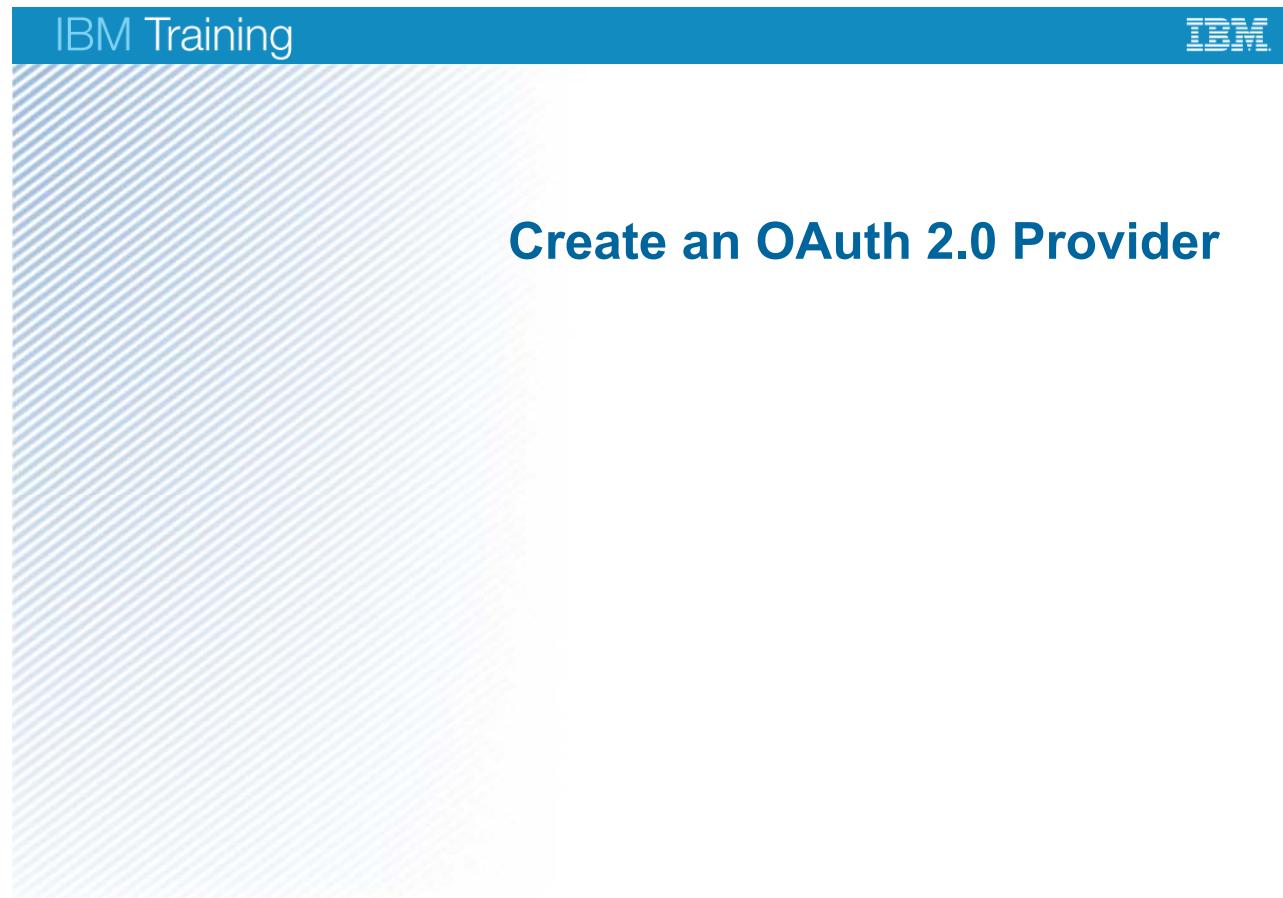


Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-5. What is an OAuth 2.0 Provider?

## 11.1. Create an OAuth 2.0 Provider



Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

*Figure 11-6. Create an OAuth 2.0 Provider*

## Topics

### Create an OAuth 2.0 Provider

- Secure an API with OAuth 2.0 authorization

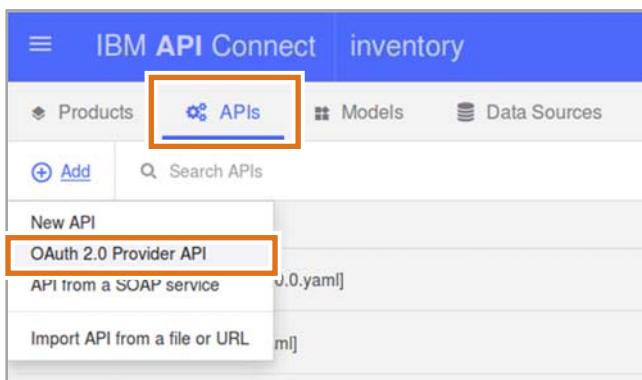
Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

*Figure 11-7. Topics*

## Step 1: Create an OAuth 2.0 Provider

1. In API Designer, select the **API** tab to list all APIs
2. Select **Add > OAuth 2.0 Provider API**



3. In the OAuth 2.0 Provider wizard, enter a title and the base path for the authorization services
4. Optionally, add the OAuth 2.0 Provider API to a product

[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

Figure 11-8. Step 1: Create an OAuth 2.0 Provider

To create an OAuth 2.0 Provider API, open the API wizard in the API Designer web application and complete the steps that are provided on the slide.

## Step 2: Configure the OAuth 2.0 Provider settings

The screenshot shows the configuration interface for an OAuth 2.0 provider. On the left, a sidebar lists various sections: Host, OAuth 2 (highlighted with a yellow circle labeled 1), Client type, Consumes, Produces, Lifecycle, Policy Assembly, Security Definitions, Security, Extensions, Properties, Paths, /oauth2/authorize, /oauth2/token, Analytics, Parameters, Definitions, access\_token\_response, and Services. The main panel is titled 'OAuth 2' and contains the following sections:

- Client type:** Set to 'Public' (highlighted with a yellow circle labeled 2).
- Scopes:** A list containing 'scope1' (highlighted with a yellow circle labeled 3).
- Description:** Contains the text 'Description 1'.
- Advanced Scope Check:** A detailed description of the feature, with two toggle buttons: 'Enable Application Scope Check' and 'Enable Owner Scope Check' (both are off).
- Grants:** A section with four checkboxes, all of which are checked: 'Implicit', 'Password', 'Application', and 'Access Code' (highlighted with a yellow circle labeled 4).

- Review the **OAuth 2.0** settings in the **OAuth 2.0 Provider API**

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-9. Step 2: Configure the OAuth 2.0 Provider settings

1. In the OAuth 2.0 Provider API definition, select the **OAuth 2** section.
2. Select either a **public** or **confidential** OAuth client type. The following slide explains the implications of each client type.
3. 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 state that a customer can view only bank accounts that the customer owns. You define the name and description for the scope.
4. In the **grants** section, select which one 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.

## OAuth 2.0 Provider API: Client types

- The OAuth 2.0 specification defines two types of clients:
  - Public
  - Confidential
- **Public clients** should not 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

Figure 11-10. OAuth 2.0 Provider API: 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.

## OAuth 2.0 Provider API: OAuth flow and grant types

### OAuth flow    OAuth grant type and explanation

Implicit	<ul style="list-style-type: none"> <li>▪ Uses an <b>implicit</b> grant type</li> <li>▪ The authorization server sends back an access token after the resource owner authorizes the client application to use the resource</li> </ul>
Password	<ul style="list-style-type: none"> <li>▪ Uses the <b>resource owner password credentials</b></li> <li>▪ The client application sends the user name and password for a user on the resource server</li> </ul>
Application	<ul style="list-style-type: none"> <li>▪ Uses the <b>client credentials</b></li> <li>▪ The client application sends its own credentials when it accesses resources under its own control, or previously arranged with the authorization server</li> </ul>
Access code	<ul style="list-style-type: none"> <li>▪ After the authorization server authenticates the resource owner, the authentication server sends back a custom redirect URI and an authorization code</li> <li>▪ The client application opens the redirect URI with the authorization code to retrieve an access token for a resource</li> </ul>

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-11. OAuth 2.0 Provider API: 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.



## Step 3: Authentication and authorization settings

Collect credentials using

Default form 5

**Authentication**

Authenticate application users using

Authentication URL 6

Authentication URL  
https://example.com/auth/url

TLS Profile

When using the Application grant type, authentication settings are not applicable and are ignored.

**Authorization**

Authorize application users using

Authenticated 7

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-12. Step 3: Authentication and authorization settings

Scroll down to reveal the authentication and authorization settings of the OAuth 2.0 Provider.

5. To authenticate the client, the authorization service must **extract** the **client identity**. You can retrieve client credentials from a web form, the HTTP Basic authentication header, or redirect to a login website.
6. The **Authentication** setting determines how the authorization service verifies the identity of the client. In this example, the OAuth 2.0 Provider API 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.
7. The **Authorization** setting determines how the OAuth provider authorizes access to resources. The **Authenticated** setting automatically grants access to any authenticated client. Otherwise, use the **custom form** or **default form** to ask the resource owner which resources to grant access.



## Step 4: Token settings

**Tokens**

Access tokens

Time to live (seconds)

3600 ← 8 ↗

Enable refresh tokens

Count

2048 ↗

Time to live (seconds)

2682000 ↗

Enable maximum consent

Maximum lifetime that the permission is valid before the application must gather consent again. Enter a value greater or equal to refresh tokens time to live. The default value is 0, which disables this feature.

Enable revocation

Use DataPower Gateway

Use a revocation URL

Revocation URL

← 9 ↗

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-13. Step 4: Token settings

Scroll down to reveal the token settings of the OAuth 2.0 Provider API settings.

8. The **Access tokens** settings determine how long to keep tokens alive and whether to enable a renewal service for access tokens.
9. The resource owner uses the **Revocation URL** setting to log out and invalidate an access token that the OAuth provider issued previously.

## OAuth 2.0 Provider API: Token settings

- Access tokens
  - To make the OAuth security scheme safer, the access tokens expire after a set amount of time
  - The client application must renew the token with the **token refresh service**
  - You can limit the maximum number of refresh tokens, and the life span of refresh tokens
- Token revocation service
  - When the client application is no longer needed, the user can **revoke** access rights to the client application
  - To revoke a token, call the **token revocation service** with the access token

[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

Figure 11-14. OAuth 2.0 Provider API: Token settings

In an OAuth 2.0 message flow, the **token** API issues access tokens to the client application. When the client application makes an API request, it sends the access token in the **authorization** header. If the token is valid, the API permits access to API operations and resources.

To make the OAuth security scheme safer, the access tokens expire after a set amount of time. The client application must renew the token with the **token refresh service**. You can limit the maximum number of refresh tokens and the life span of refresh tokens.

The **token revocation service** marks an access token as invalid. When the client application finishes its API requests, it can log out and revoke access rights from the tokens. If any party suspects that the tokens are compromised, they can revoke tokens. To revoke a token, call the **token** revocation service with the access token.

## OAuth 2.0 Provider: Metadata

- In some scenarios, you want to store application or network-specific information when the OAuth provider generates an access token
- For example, the **authorization** and **metadata** API operations can save:
  - Information about the resource owner in the access token
  - The grant type that was used to obtain the access token
  - A custom confirmation code that the provider API returns along with the access token
- When you request an access token from the provider API, you specify the location in which you want to save the metadata
  - Store metadata in the response message, **along** with the access token
  - Encode metadata **inside** the access token

[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

*Figure 11-15. OAuth 2.0 Provider: Metadata*

In some scenarios, you want to store application or network-specific information when the OAuth provider generates an access token.

For example, the authorization and metadata API operations can save information about the resource owner in the access token and the grant type that was used to obtain the access token. They can also save a custom confirmation code that the provider API returns along with the access token.

When you request an access token from the provider API, you specify the location in which you want to save the metadata.

You can store metadata in the response message, along with the access token. Alternatively, you can encode metadata inside the access token.

## Example: Metadata that is stored in OAuth response message

- When you call the **authorization URL**, the provider API can insert information in a field that is named **metadata** within the response message body

```
{
  "token_type": "bearer",
  "access_token": "AAEkNzhjDHY...w7ZU",
  "metadata": "m:custom-metadata-stored-in-payload",
  "expires_in": 3600,
  "scope": "read",
  "refresh_token": "AAEnjSynCM...HdNg"
}
```

- Advantages:**
  - The client application can easily read the token metadata
- Disadvantages:**
  - The client might leave out the metadata when it saves the access token

Figure 11-16. Example: Metadata that is stored in OAuth response message

When you call the **authorization URL**, the provider API can insert information in a field that is named **metadata** within the response message body. The advantage of this scheme is that the client application can easily read the token metadata. The disadvantage of this scheme is that the client might inadvertently leave out metadata when it saves the access token.

## Example: Metadata that is stored in OAuth access token

- To decode metadata that is stored in the access token, call the token introspection endpoint

```
{
  "active": true,
  "token_type": "bearer",
  "client_id": "b91e945a-21wf-4869-bb7bay130d",
  "username": "Student",
  "exp": 1485811873,
  "expstr": "2017-01-30T21:31:13Z",
  "scope": "read",

  "miscinfo", "m:custom-metadata-encoded-in-access-token",

  "client_name": "Inventory-client-app"
}
```

- Advantages:**

- When the client saves the access token, it also saves the token metadata

- Disadvantages:**

- The client must send the token to the introspection endpoint to read the metadata

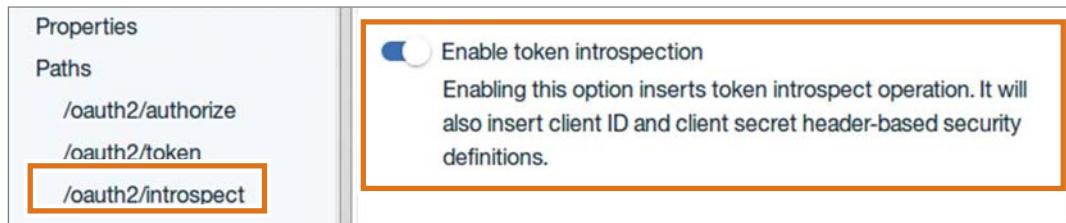
Figure 11-17. Example: Metadata that is stored in OAuth access token

To decode metadata that is stored in the access token, call the token introspection endpoint. The advantage of this scheme is that the metadata exists within the access token. When the client saves the token, it also captures the metadata.

The disadvantage of this scheme is that the client application cannot read the metadata directly: it must make an API request to the token introspection endpoint to read the metadata.

## How to: Add a token introspection endpoint

- To add the **token introspection** service to your OAuth 2.0 Provider, select **Enable token introspection** from the OAuth 2.0 Provider settings



- The **API Designer** editor adds a third API operation to the provider API: **/oauth2/introspect**
  - You can rename the path in the OAuth 2.0 Provider API definition editor

Figure 11-18. How to: Add a token introspection endpoint

## How to: Add a metadata URL to the OAuth 2.0 Provider

- The **metadata URL** is an external service that tracks token metadata
  - The OAuth 2.0 Provider does not implement this service: it merely stores the network address of an external metadata service
- When you call the **authorization URL** in the OAuth 2.0 Provider, the gateway calls the **metadata URL** and overwrites the metadata fields in the token
- If you do not want to use an external metadata service, leave this field blank
  - The **authorization** service in the OAuth 2.0 Provider sets and retrieves metadata for your access tokens

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-19. How to: Add a metadata URL to the OAuth 2.0 Provider

## Example: Specify an external OAuth metadata URL

**Metadata**

Metadata can be requested from a remote server with both the Authentication URL and the Metadata URL. The Authentication URL must be configured for "Authentication". Metadata URL can be specified in the field below. Both URLs can return either of the two metadata response headers and the header value is placed according to the header name. The response header "API-OAUTH-METADATA-FOR-ACCESSTOKEN" value is placed within the access token as "miscinfo". The response header "API-OAUTH-METADATA-FOR-PAYOUTLOAD" value is placed in the response payload as "metadata".

**Metadata URL**

[Redacted]

**TLS Profile**

[Redacted]

- For more information about **OAuth metadata**, see [https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.toolkit.doc/con\\_metadata.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/con_metadata.html)

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-20. Example: Specify an external OAuth metadata URL

## 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 2017

*Figure 11-21. Secure an API with OAuth 2.0 authorization*

## Topics

- Create an OAuth 2.0 Provider
- ▶ Secure an API with OAuth 2.0 authorization

[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

*Figure 11-22. Topics*



## Step 1: Create an OAuth 2.0 security definition

To secure your API with OAuth 2.0, you must create an OAuth 2.0 security definition and apply it to your API

1. In API Designer, click the **APIs** tab
2. Select an existing API definition or create a new API definition

A screenshot of the API Designer interface. At the top, there are tabs: "Products", "APIs" (which is highlighted with an orange border), "Models", and "Data Sources". Below the tabs is a search bar with "Add" and "Search APIs" buttons. The main area shows a table with two rows. The first row contains the title "example-oauth-provider 1.0.0 [example-oauth-provider\_1.0.0.yaml]". The second row, which is highlighted with an orange border, contains the title "pricing 1.0.0 [pricing.yaml]". Both rows have a small "Edit" icon to their left.

[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

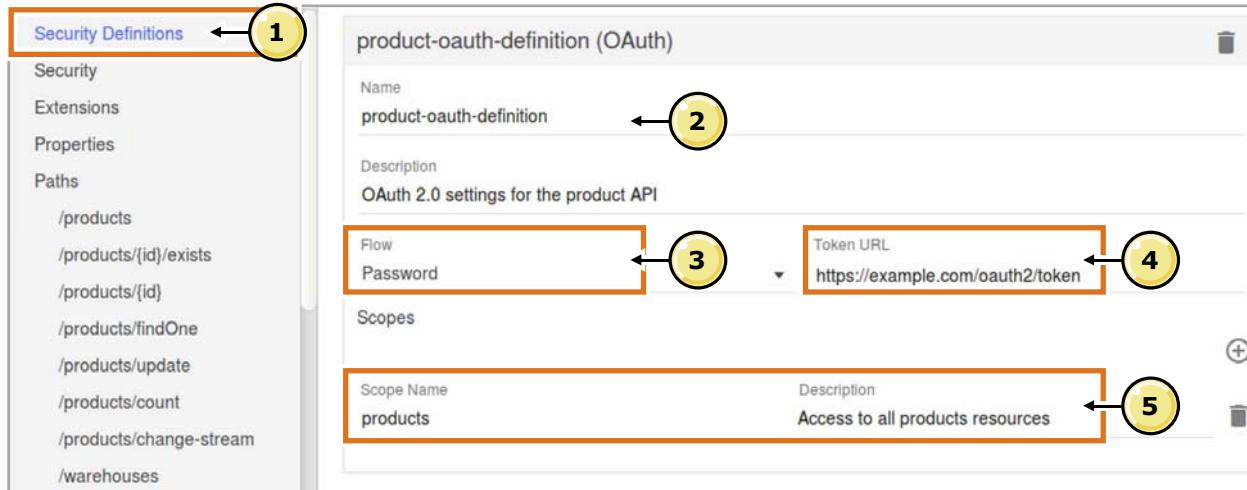
Figure 11-23. Step 1: Create an OAuth 2.0 security definition

Recall that the OAuth 2.0 Provider API is the authorization server in the OAuth 2.0 message flow. The suggested practice is to keep the OAuth 2.0 Provider API separate from your business APIs. In the example on this slide, you want to enforce OAuth 2.0 authorization in the **pricing** API operations.

To secure your API, you must create an OAuth 2.0 security definition and apply it to the API definition. First, click the **APIs** tab in the API Designer web application. Open an existing API definition, or create an API definition.

## Step 2: Create an OAuth 2.0 security definition

1. Select the **Security Definitions** section in your API definition
2. Select **New > OAuth** to create an OAuth 2.0 security definition



Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-24. Step 2: Create an OAuth 2.0 security definition

The OAuth 2.0 security definition states how API Connect authenticates and authorizes clients to your API.

1. In the API definition, select the **Security Definitions** section.
2. Select **New > OAuth** to create an OAuth security definition. Provide a name and description for the security definition.
3. Select which type OAuth flow to enforce on the API. The OAuth flow that you choose must be a type that your OAuth 2.0 Provider supports.
4. In this example, you support the **resource owner client credentials** grant type. Specify a **Token URL** in which you exchange the client credentials for an access token.
5. Optionally, specify an OAuth scope. In this example, authenticated users can access all APIs in the product scope.

## Unit summary

- Explain the concept of an OAuth provider
- Identify the role of the authorization server in the OAuth flow
- Identify the OAuth 2.0 roles, client types, and schemes
- Explain the difference between public and confidential schemes
- Explain the OAuth flow and grant types
- Explain the concept of access tokens
- Explain the concept of the token revocation service
- Explain the purpose of the token introspection endpoint
- Explain the concept of OAuth metadata
- Explain the relationship between the OAuth provider and the API security definition

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-25. Unit summary

## Review questions

1. In API Connect, which OAuth grant type setting represents the user name and password of the client application, instead of the application user?
  - A. Implicit
  - B. Password
  - C. Application
  - D. Access code
  
2. Which one of the following statements best describe the concept of OAuth 2.0 clients?
  - A. Confidential clients do not send a client ID.
  - B. Public clients do not send a client ID.
  - C. Confidential clients do not ensure that they can keep a client password secret.
  - D. Public clients do not ensure that they can keep a client password secret.



Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-26. Review questions

Write your answers here:

- 1.
  
- 2.

## Review answers

1. In API Connect, which OAuth grant type setting represents the user name and password of the client application, instead of the application user?
  - A. Implicit
  - B. Password
  - C. Application
  - D. Access code

The answer is C.
  
2. Which one of the following statements best describe the concept of OAuth 2.0 clients?
  - A. Confidential clients do not send a client ID.
  - B. Public clients do not send a client ID.
  - C. Confidential clients do not ensure that they can keep a client password secret.
  - D. Public clients do not ensure that they can keep a client password secret.

The answer is D.

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-27. Review answers



## Exercise: Declaring an OAuth 2.0 Provider and security requirement

Creating an OAuth 2.0 Provider

© Copyright IBM Corporation 2017

Figure 11-28. Exercise: Declaring an OAuth 2.0 Provider and security requirement

## Exercise objectives

- Define an OAuth 2.0 Provider API in the API Designer web application
- Configure the client ID and client secret security definition
- Declare and enforce an OAuth 2.0 security definition with the API Designer web application



[Creating an OAuth 2.0 Provider](#)

© Copyright IBM Corporation 2017

*Figure 11-29. 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. Current versions of Docker include a swarm mode that is used for scalability and failover in distributed server runtime environments. You learn how to deploy API implementations to a Docker image and a Docker swarm.

## 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
- Describe the Docker swarm mode for managing a cluster of Docker engines

## What is Docker?

- Docker is an open source software technology to easily create



lightweight,  
portable,  
self-sufficient

containers from any application

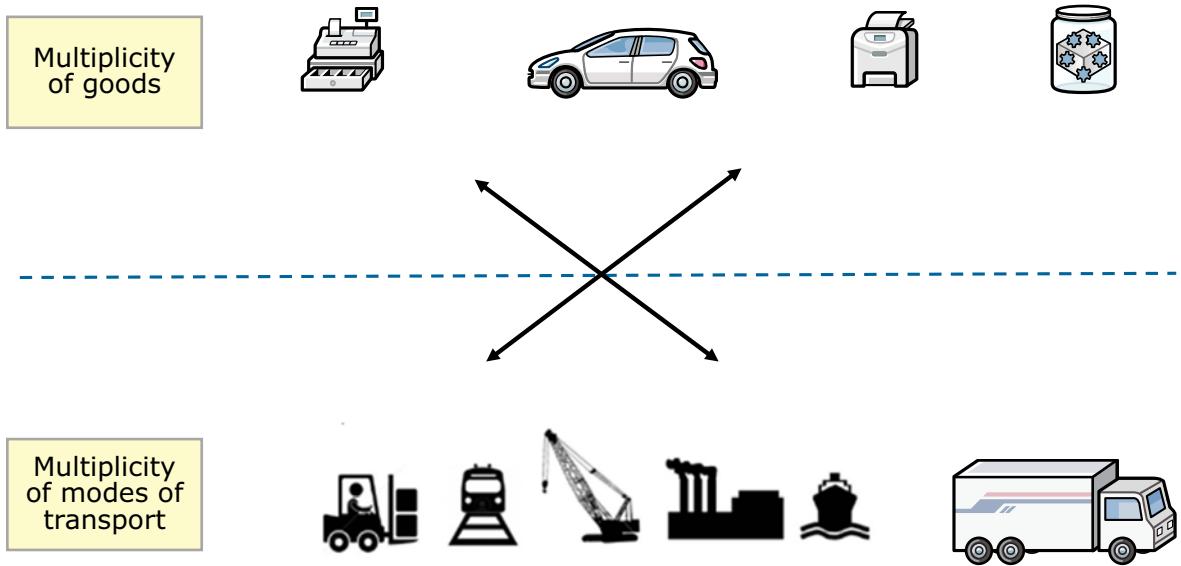
Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-2. What is Docker?

## Cargo transport pre-1960

- Challenges of shipping things



Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-3. Cargo transport pre-1960

The slide depicts some of the challenges of shipping goods before 1960.

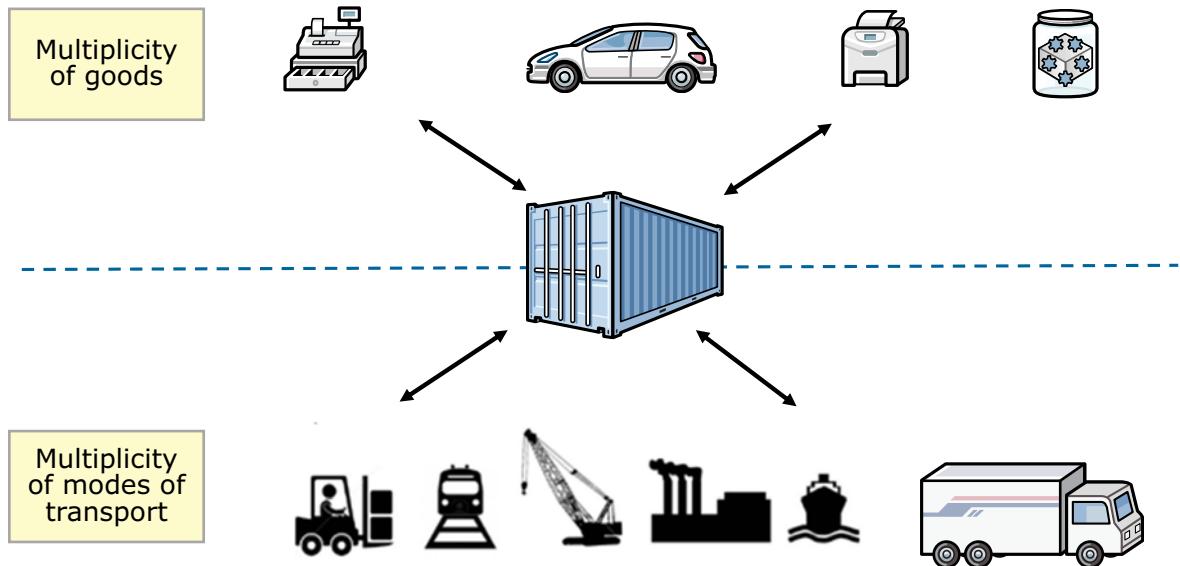
How do you safely and efficiently transport various goods by using different modes of transport?

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 2017

Figure 12-4. Solution: Cargo container

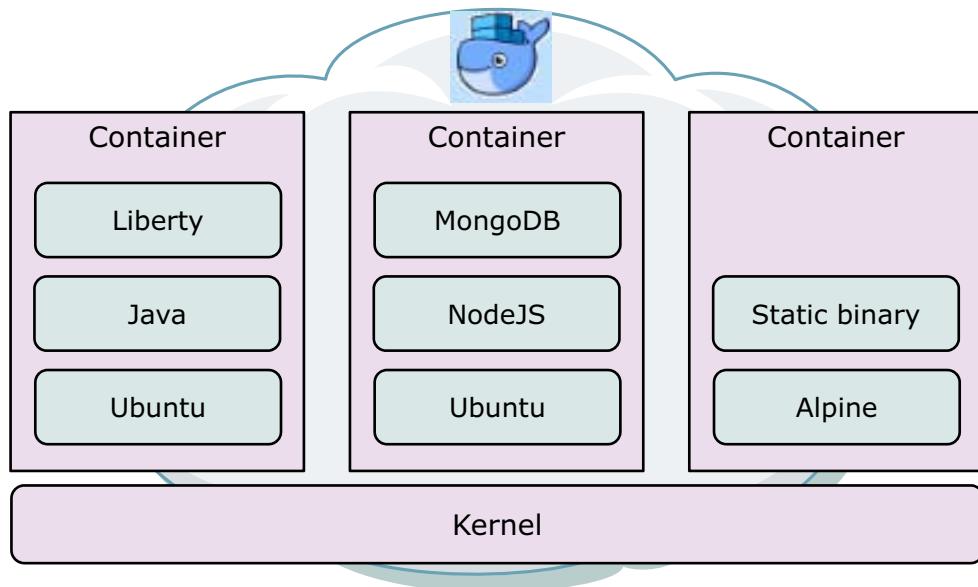
With the cargo container, you can transport a multiplicity of goods.

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 2017

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 2017

*Figure 12-6. Why use Docker containers?*

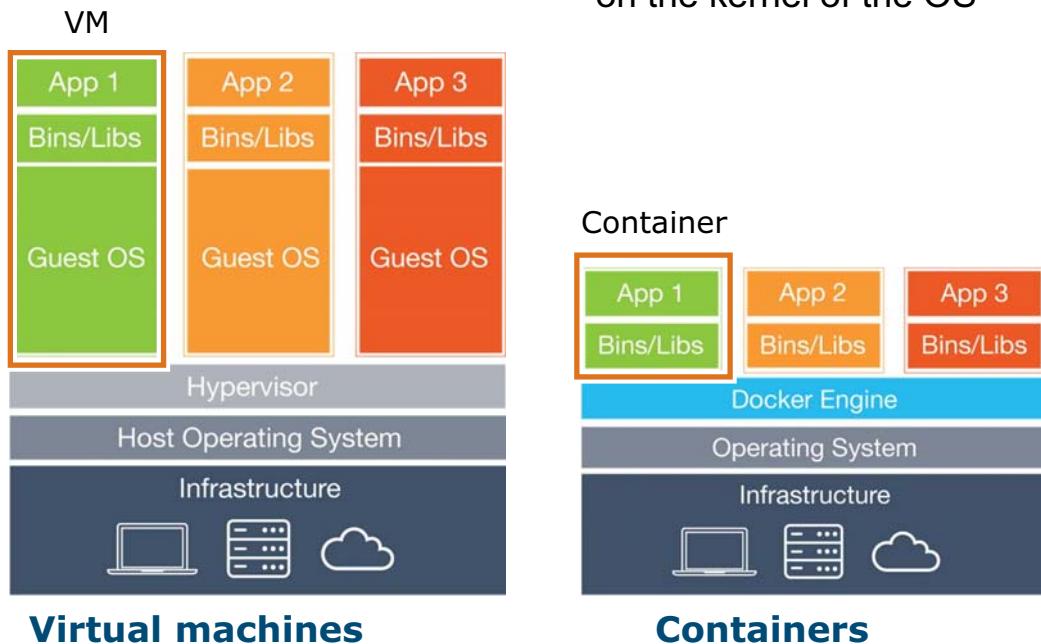
Using containers to run applications is beneficial in many ways.

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 2017

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

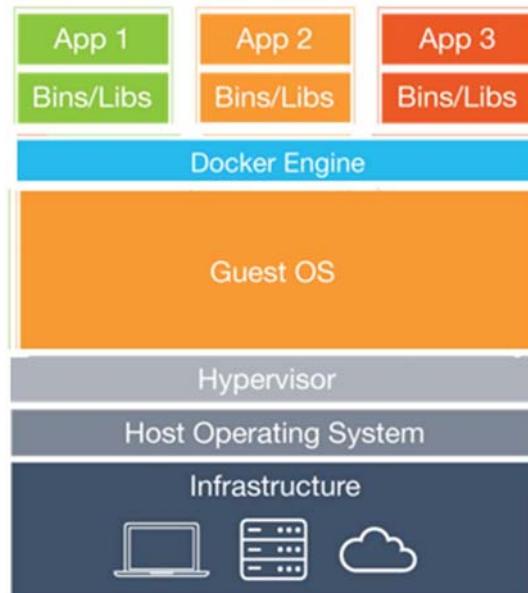
- A hypervisor is a function that abstracts – isolates – operating systems and applications from the underlying computer hardware
- The abstraction allows the underlying host hardware to independently operate one or more virtual machines as guests
  - In this manner, multiple guest VMs can effectively 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 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 2017

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 2017

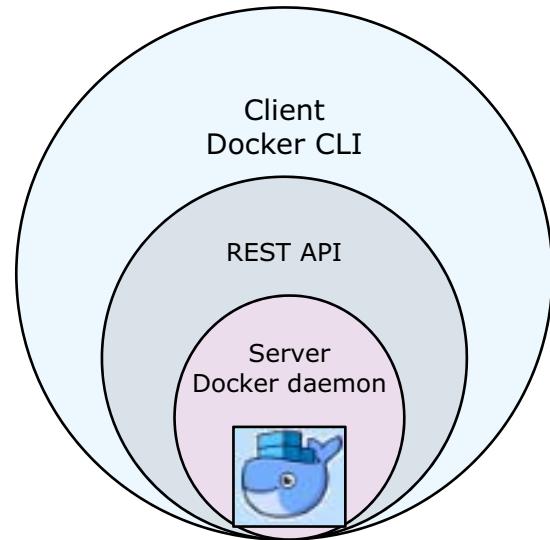
Figure 12-10. Docker terminology

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 issue 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 2017

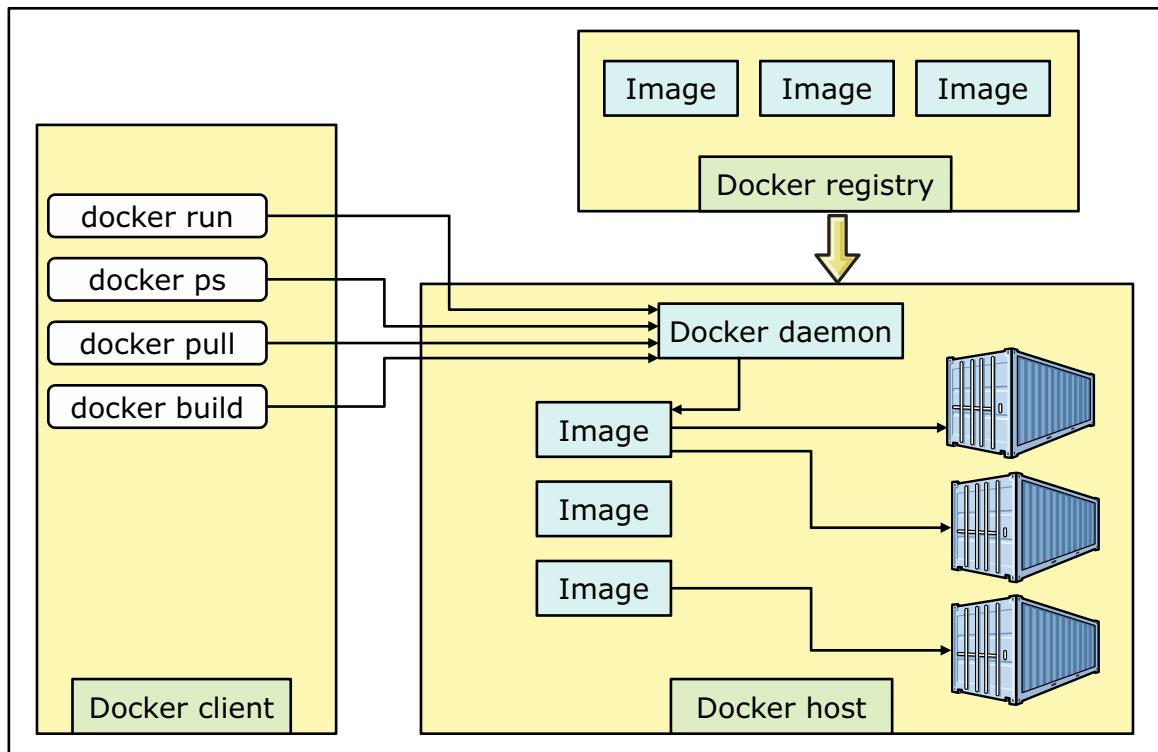
Figure 12-11. Docker Engine

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 2017

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

```
student@xubuntu-vm:~$ docker version
Client:
  Version:      17.03.1-ce
  API version:  1.27
  Go version:   go1.7.5
  Git commit:   c6d412e
  Built:        Mon Mar 27 17:10:36 2017
  OS/Arch:      linux/amd64

Server:
  Version:      17.03.1-ce
  API version:  1.27 (minimum version 1.12)
  Go version:   go1.7.5
  Git commit:   c6d412e
  Built:        Mon Mar 27 17:10:36 2017
  OS/Arch:      linux/amd64
  Experimental: false
student@xubuntu-vm:~$
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-13. Docker version

This page shows the output from running the `docker version` CLI command. The output shows which versions of the Docker client and server are installed and also the REST API version that is used to communicate with the Docker daemon.

## Docker CLI

- Example `docker run` command

```
localuser@manager1:~$ 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
    which 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@manager1:~$
```

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 2017

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 2017

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
```

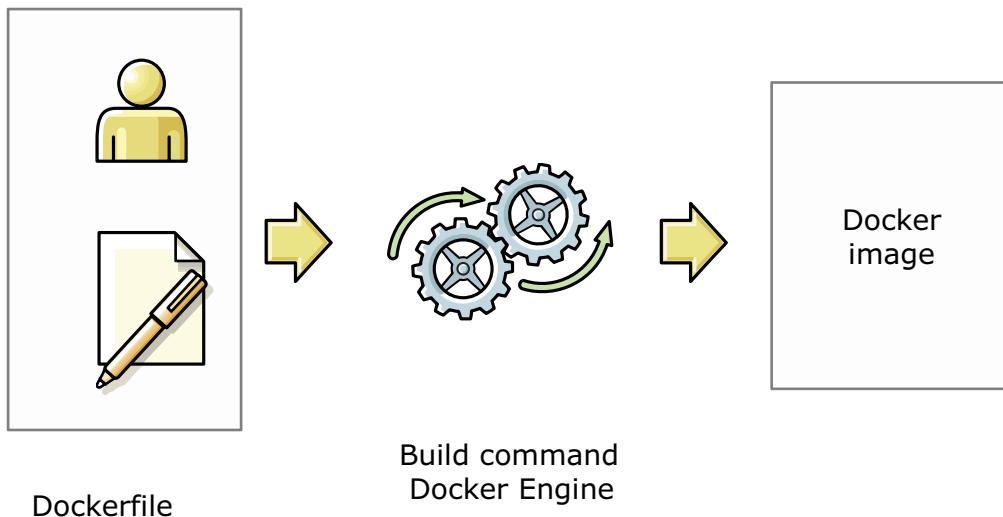
Figure 12-17. Dockerfile

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 executes 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 2017

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:3.6
RUN apk add --update nodejs nodejs-npm && npm install npm@4.4.4 -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 mysql.think.ibm:192.168.225.10 --add-host mongo.think.ibm:192.168.225.10 -p 3000:3000 apic-inventory-image
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

*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:

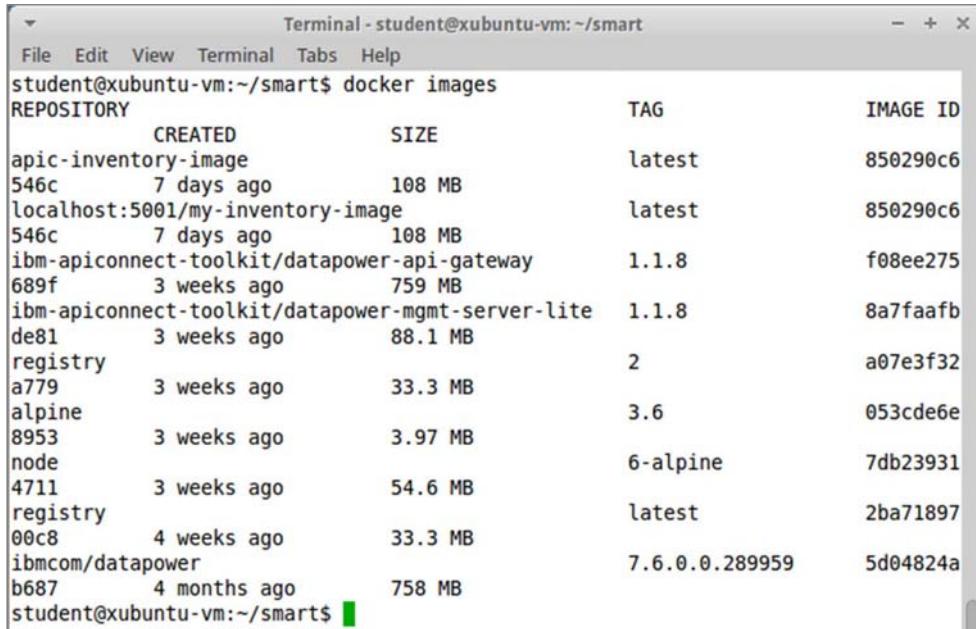
```
docker build -t <image-name> <path>
```

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



```
Terminal - student@xubuntu-vm:~/smart
File Edit View Terminal Tabs Help
student@xubuntu-vm:~/smart$ docker images
REPOSITORY          CREATED             SIZE        TAG      IMAGE ID
apic-inventory-image 546c    7 days ago   108 MB    latest   850290c6
localhost:5001/my-inventory-image 546c    7 days ago   108 MB    latest   850290c6
ibm-apiconnect-toolkit/datapower-api-gateway 689f    3 weeks ago  759 MB    1.1.8    f08ee275
ibm-apiconnect-toolkit/datapower-mgmt-server-lite de81    3 weeks ago  88.1 MB
registry              a779    3 weeks ago  33.3 MB
alpine                8953    3 weeks ago  3.97 MB
node                  4711    3 weeks ago  54.6 MB    6-alpine  7db23931
registry              00c8    4 weeks ago  33.3 MB    latest   2ba71897
ibmcom/datapower      b687    4 months ago 758 MB    7.6.0.0.289959  5d04824a
student@xubuntu-vm:~/smart$
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

*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 mysql.think.ibm:192.168.225.10 --add-host
mongo.think.ibm:192.168.225.10 -p 3000:3000 apic-inventory-image

> inventory@1.0.0 start /inventory
> node .

Web server listening at: http://localhost:3000
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

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

```
Terminal - student@xubuntu-vm:~/smart
File Edit View Terminal Tabs Help
student@xubuntu-vm:~/smart$ docker ps
CONTAINER ID        IMAGE               CREATED             STATUS              PORTS
CMD                CREATED
NAMES
38bfcd6335ca      ibm-apiconnect-toolkit/datapower-api-gateway:1.1.8   "/bin/drouter"   About a minute ago   Up About a minute   0.0.0.0:32775->80/
tcp, 0.0.0.0:32774->5554/tcp, 0.0.0.0:32773->9090/tcp, 0.0.0.0:4001->9443/tcp
smart_datapower-api-gateway_1
9a1560977dc1      ibm-apiconnect-toolkit/datapower-mgmt-server-lite:1.1.8 "node lib/server.js"   About a minute ago   Up About a minute   0.0.0.0:32772->2443/tcp
smart_datapower-mgmt-server-lite_1
student@xubuntu-vm:~/smart$
```

- docker ps -a shows all containers, including containers that are not running

```
Terminal - student@xubuntu-vm:~
File Edit View Terminal Tabs Help
student@xubuntu-vm:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND            CREATED             STATUS              NAMES
          STATUS        PORTS
499659843f13      apic-inventory-image   "npm start"        4 days ago         Exited (0) 4 days ago   quirky_yallow
47a4c3206d92      apic-inventory-image   "npm start"        5 days ago         Exited (0) 4 days ago   zen_kalam
student@xubuntu-vm:~$
```

Figure 12-22. Listing containers

The docker ps command can be used to inspect running or stopped 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 if running multiple containers

```
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`

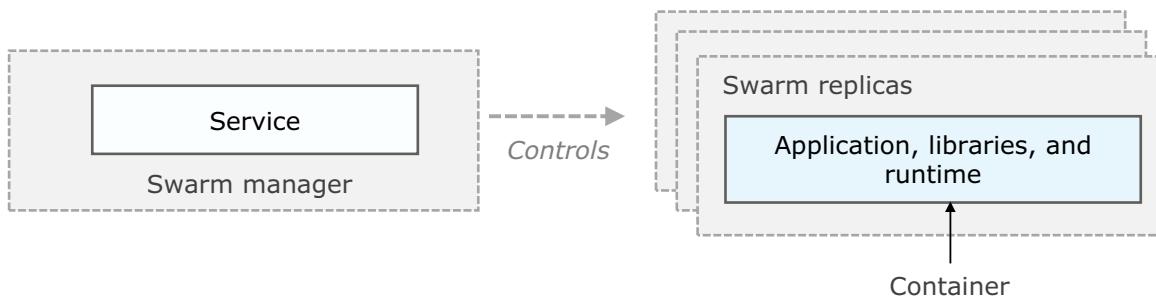
Run `docker <command> --help` for more information on each command.

This page shows examples of docker commands.

For a complete list of the docker CLI commands, see the Reference section of the online Docker documentation at [docs.docker.com](https://docs.docker.com), or type `docker -help` from the terminal of a machine where Docker is installed.

## Terminology: Service and swarm

- You can use a **service** to scale containers across multiple Docker daemons, which all work together as a **swarm** with multiple managers and workers.
  - With the service, you can define the state that you want, such as the number of replicas of the service that must be available
  - To the consumer, the Docker service appears to be a single application
- Each member of a swarm (node) is a Docker daemon
- Docker Engine supports swarm mode in Docker 1.12 and higher



Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-24. Terminology: Service and swarm

To deploy an application image when Docker Engine is in swarm mode, you create a service. The swarm manager accepts the service definition as the preferred state for the service. The swarm manager schedules the service on the nodes in the swarm as one or more replica tasks.

Each of the replica instances is a task in the swarm. The container is an instantiation of the task. As soon as the container is active, the scheduler recognizes that the task is in a running state. If the container terminates, the task terminates.

For more information, see the Docker documentation at

<https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/>

With the Docker swarm technology, you can publish your applications to a distributed, clustered containerized environment.

A swarm manager administers a cluster of Docker containers. At run time, the swarm manager routes API requests to a Docker cluster member that runs an instance of your API and its runtime dependencies.

You can use a service to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers.

## Define the service

- When you create the service, you specify which image to use and the options for the service

```
version: "3"
services:
  inv-service:
    # replace repo:tag with your name and image
    image: localhost:5000/my-inventory-image
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure
    extra_hosts:
      - "mysql.think.ibm:192.168.225.10"
      - "mongo.think.ibm:192.168.225.10"
    ports:
      - "3000:3000"
    networks:
      - webnet
networks:
  webnet:
```

**docker-compose.yml**

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-25. Define the service

When you create a service, you specify which container image to use and which commands to execute inside running containers. You also define options for the service, including:

- The port where the swarm makes the service available outside the swarm
- An overlay network for the service to connect to other services in the swarm
- CPU and memory limits and reservations
- A rolling update policy
- The number of replicas of the image to run in the swarm

You create a definition of a service in a Docker compose file.

A YAML file with the syntax to create the service is displayed on the page.

The Docker compose file is a text file in YAML format that defines services, networks, and volumes.

A service definition contains configuration that is applied to each container that is started for that service, similar to passing command-line parameters to the docker run command.

## Example of initializing a Docker swarm

- Start Docker in swarm mode

```
$ docker swarm init --advertise-addr 192.168.225.10

Swarm initialized: current node (7m8qit3gff8ovwd7pzoepy83c) is
now a manager.
To add a worker to this swarm, run the following command:
  docker swarm join \
    --token SWMTKN-1-
53ej1ky1dk7cucw0zt1xhonvossc99k1nq4g2n3ml1v6qseuj-
0xs5ctx9nu173qg10qg4d7h1y \
  192.168.225.10:2377

To add a manager to this swarm, run 'docker swarm join-token
manager' and follow the instructions.
```

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-26. Example of initializing a Docker swarm

The first step in running a service on a Docker swarm is to initialize the swarm. When you run the command to create a swarm, the Docker engine starts running in swarm mode. Run the command `docker swarm init` to create a single-node swarm on the current node. Swarm mode is enabled, and the current node is designated as the leader manager for the swarm.

Next, you create a service to run on the Docker swarm mode.

## Deploy the service to a Docker swarm

- Deploy the service that is defined in the YML file to a named stack
- Usage: `docker stack deploy [OPTIONS] STACK`
- Options: `-c`, `--compose-file` string: Path to a compose file

```
$ docker stack deploy -c docker-compose.yml mystack
```

```
Creating network mystack_webnet
Creating service mystack_inv-service
```

- List the running services

```
$ docker services ls
```

ID	NAME	MODE	REPLICAS	IMAGE
ns2z64c7a16m	mystack_inv-service	replicated	2/2	localhost:5001/my-inventory-image:latest

*Figure 12-27. Deploy the service to a Docker swarm*

You defined the service in a YAML file that was shown earlier.

Now, you deploy the service to a stack in Docker swarm mode.

In this example, you deploy the service that was defined in the `docker-compose.yml` file in an earlier slide.

The `docker stack deploy` command deploys a complete application stack to the swarm.

A stack is a group of interrelated services that share dependencies and can be controlled and scaled together. You can deploy a single service stack that is running on a single host, but that typically is not what takes place in production. Instead, you make multiple services relate to each other, and then run them on multiple machines.

In the lab exercise that follows this unit, you run multiple replicas of a service on a single stack on a single host machine.

## 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
- Describe the Docker swarm mode for managing a cluster of Docker engines

[Deploying an API to a Docker container](#)

© Copyright IBM Corporation 2017

*Figure 12-28. 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. To deploy an application image when Docker Engine is in swarm mode, you create which of the options that are listed?
  - A. Container
  - B. Task
  - C. Service
  - D. Replica



Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-29. 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. To deploy an application image when Docker Engine is in swarm mode, you create which of the options that are listed?

- A. Container
- B. Task
- C. Service
- D. Replica

The answer is C.

## Exercise: Deploying an API implementation to a container runtime

Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-31. Exercise: Deploying 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
- Build a Docker image by using the docker command with the Dockerfile
- Verify that the API runs on the Docker image
- Push the image to a local registry
- Initialize the Docker swarm
- Use the Docker stack command to deploy a swarm service
- Test that the API runs on the swarm service



Deploying an API to a Docker container

© Copyright IBM Corporation 2017

Figure 12-32. Exercise objectives

---

# Unit 13. Staging, publishing, and deploying an API product

## 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. It also 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

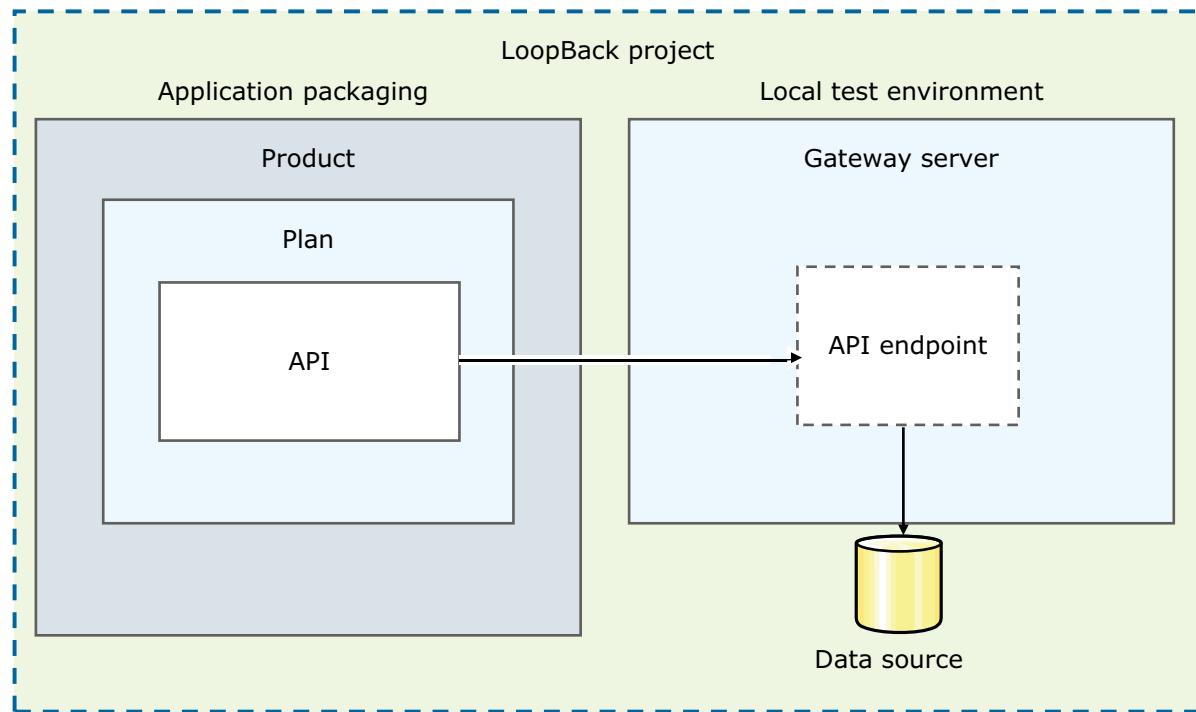
- Explain the concept of a plan, a product, and a catalog
- Explain the staging and publishing API lifecycle stages
- Define an API product and a plan
- Define product categories
- Stage a product to a catalog

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

*Figure 13-1. Unit objectives*

## Product, plan, API hierarchy



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-2. Product, plan, API hierarchy

Products, plans, and APIs are all represented in YAML files within a LoopBack project.

Testing on the workstation:

You can test your API endpoints from the API Designer explorer by starting the Gateway server and the application and then running the test client of the API Designer toolkit.

Select the API REST operation that you want to test. Then, call the operation with the test client. Depending on the type of policies, the test Gateway server can be the Micro gateway or the embedded DataPower gateway that runs in a Docker container.

## 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

[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

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

```
product: '1.0.0'
info:
  name: hello-world
  title: hello-world
  version: 1.0.0
apis:
  'hello-world':
    $ref: hello-world.yaml
visibility:
  view:
    type: public
  subscribe:
    type: authenticated
plans:
  default:
    title: Default Plan
    description: Default Plan
    approval: false
    rate-limit:
      value: 100/hour
      hard-limit: false
```

[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

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 image shows the IBM API Connect interface. On the left, there's a sidebar with tabs for 'Products' and 'APIs'. Below the tabs are buttons for '+ Add' and 'Search products'. A 'New Product' button is also present. A yellow circle labeled '1' points to the '+ Add' button. In the center, a modal window titled 'Add a new product' is open. It has fields for 'Title' (set to 'aProduct'), 'Name' (set to 'aProduct'), and 'Version' (set to '1.0.0'). There are 'Cancel' and 'Add' buttons at the bottom right. A yellow circle labeled '2' points to the 'Add' button. On the right, a detailed view of the product is shown in a tabbed interface. The 'Design' tab is selected, showing sections for 'Info', 'Contact', 'License', 'Terms of Service', 'Visibility', 'APIs', 'Plans', and 'Default'. A yellow circle labeled '3' points to the 'Default' tab. The 'Info' section on the right shows the same product details: Title 'aProduct', Name 'aProduct', Version '1.0.0', and a 'Description' field.

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

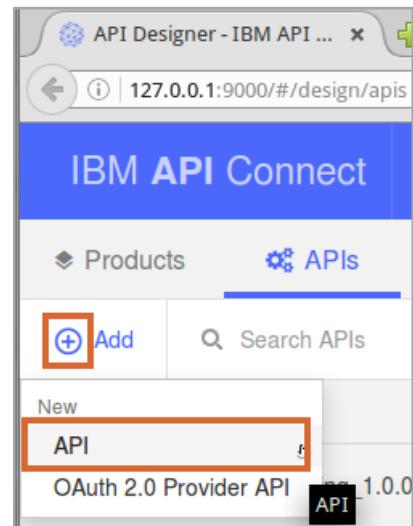
Figure 13-5. Add a product

Type `apic edit` to open the API Designer.

1. Click the **Products** tab, then click the **Add** icon.
2. Type a title, name, and version for the product in the “Add a new product” window. Click **Add**.
3. Edit the remaining fields in the Design view. A Default plan is added with 100 calls per hour.

## Add an API

- Ways to create APIs:
  - Add an API from the APIs tab of API Designer
  - Import an existing Swagger file



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

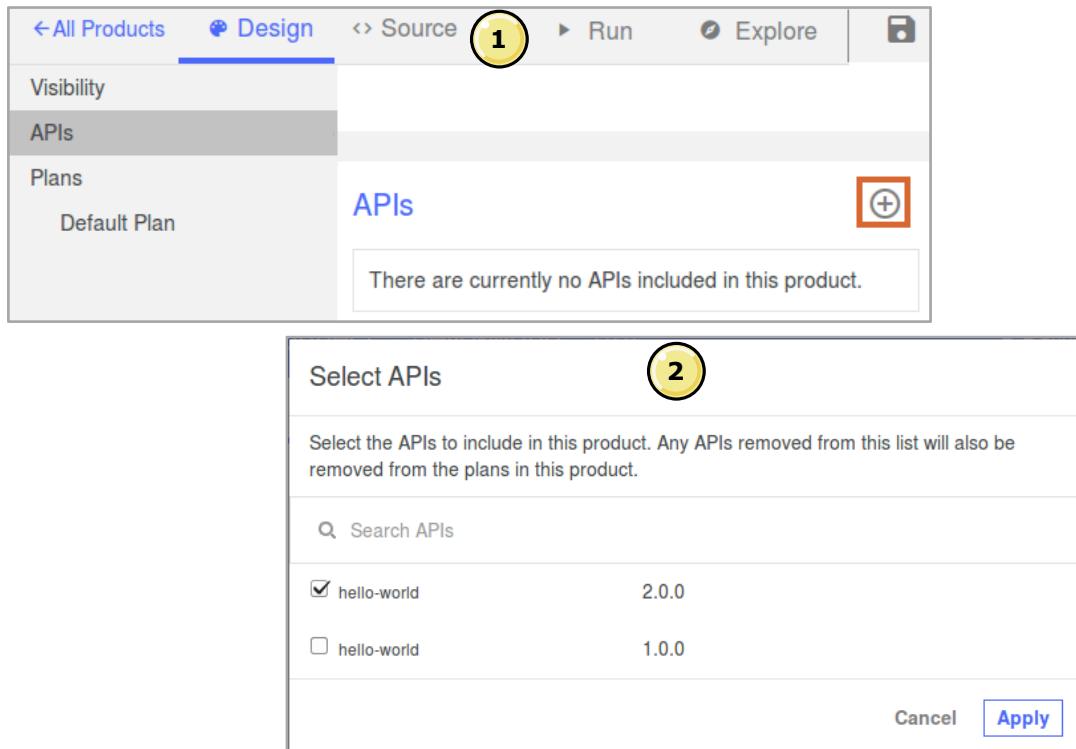
Figure 13-6. Add an API

You can add an API to a product when creating an API with the API Designer, or you can add APIs to a product later. You can also generate a Loopback API from the terminal with the apic loopback command.

# IBM Training



## Add existing APIs to a product



The screenshot shows the IBM API Designer interface. At the top, there are tabs: 'All Products', 'Design' (which is highlighted with a blue underline), 'Source', 'Run', and 'Explore'. Below the tabs, there's a sidebar with sections for 'Visibility', 'APIs' (which is also highlighted with a grey background), and 'Plans'. Under 'Plans', there's a 'Default Plan'. The main area is titled 'APIs' and contains a message: 'There are currently no APIs included in this product.' To the right of this message is a red-bordered 'Add' icon (a plus sign). In the center of the main area, there's a yellow circle with the number '1'. In the bottom right corner of the main area, there's another yellow circle with the number '2', which points to a modal dialog box titled 'Select APIs'. This dialog box has a sub-instruction: 'Select the APIs to include in this product. Any APIs removed from this list will also be removed from the plans in this product.' It includes a search bar labeled 'Search APIs' and a list of APIs. The first item, 'hello-world 2.0.0', has a checked checkbox. The second item, 'hello-world 1.0.0', has an unchecked checkbox. At the bottom right of the dialog are 'Cancel' and 'Apply' buttons.

Staging, publishing, and deploying an API product

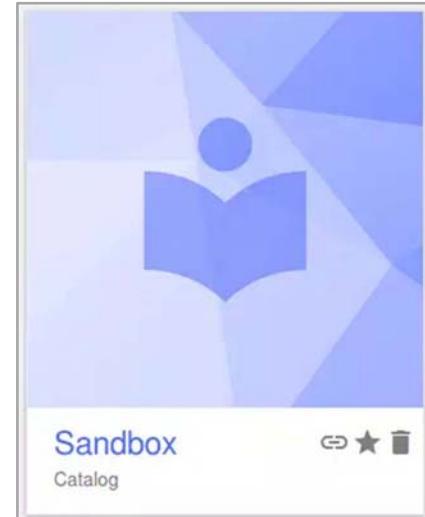
© Copyright IBM Corporation 2017

Figure 13-7. Add existing APIs to a product

1. In the API Designer, open the product from the **Products** tab.
2. Select the **APIs** tab.
3. Click the **Add** icon. Select from the list of available APIs.
4. Then, click **Apply**.
5. Save the product.

## 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



[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

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.

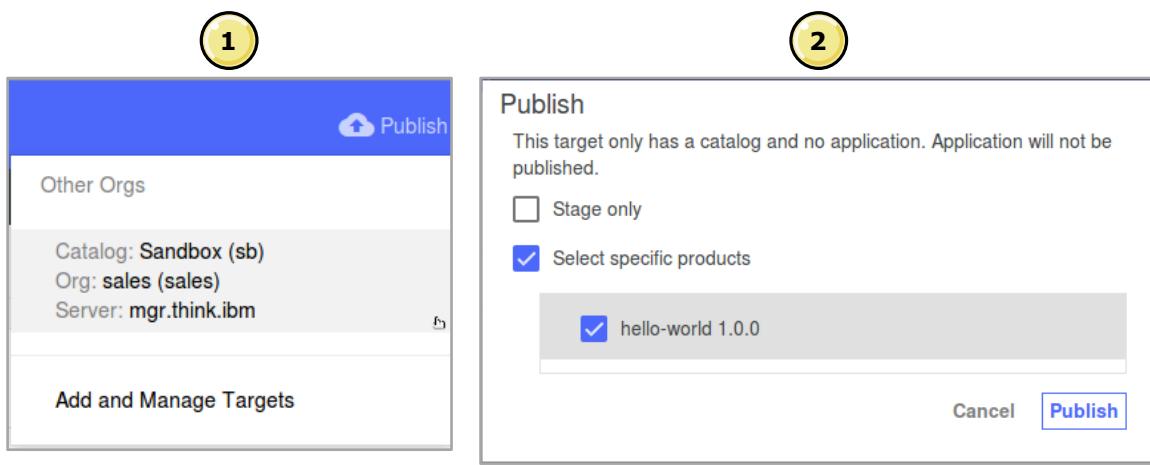
Catalogs are usually configured and managed from the API Manager user interface.



## Publish a plan and API

### Publish from the API Designer

- Select the product with the APIs to be published
- Click **Publish**
- Select the target catalog, organization, and server
- Optionally choose stage only, specific products, or both
  - Then, click **Publish**



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-9. Publish a plan and API

## Publish a plan and API result

- Successfully published products message in API Designer

**Success** Successfully published products

- Text is displayed in the terminal

```
Terminal - student@xubuntu-vm:~/hello-world
File Edit View Terminal Tabs Help
student@xubuntu-vm:~$ cd hello-world
student@xubuntu-vm:~/hello-world$ apic edit
Express server listening on http://127.0.0.1:9000
Found 1 files to publish.
Logged into mgr.think.ibm successfully
Successfully logged in for product publish
Staged /home/student/hello-world/definitions/hello-world-product.yaml to sales:sb [hello-world:1.0.0]
Published /home/student/hello-world/definitions/hello-world-product.yaml to sales:sb [hello-world:1.0.0]
Successfully published products
```

Figure 13-10. Publish a plan and API result

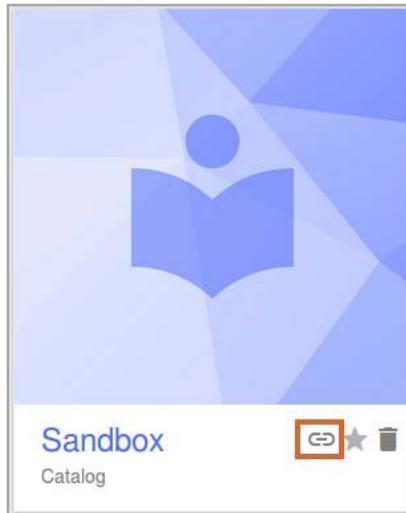
The command-line terminal displays the progress of the publish step that is made in the API Designer.

### 1+1=2 Example

```
student@xubuntu-vm:~$ cd hello-world
student@xubuntu-vm:~/hello-world$ apic edit
Express server listening on http://127.0.0.1:9000
--- Product is published from the API Designer UI ---
Found 1 files to publish.
Logged into mgr.think.ibm successfully
Successfully logged in for product publish
Staged /home/student/hello-world/definitions/hello-world-product.yaml to sales:sb [hello-world:1.0.0]
Published /home/student/hello-world/definitions/hello-world-product.yaml to sales:sb [hello-world:1.0.0]
Successfully published products
```

## Publish from the terminal: Preparation (1 of 2)

- In API Manager, open the visual display of the catalog that you want to publish to



- Click the link icon in the catalog

Staging, publishing, and deploying an API product

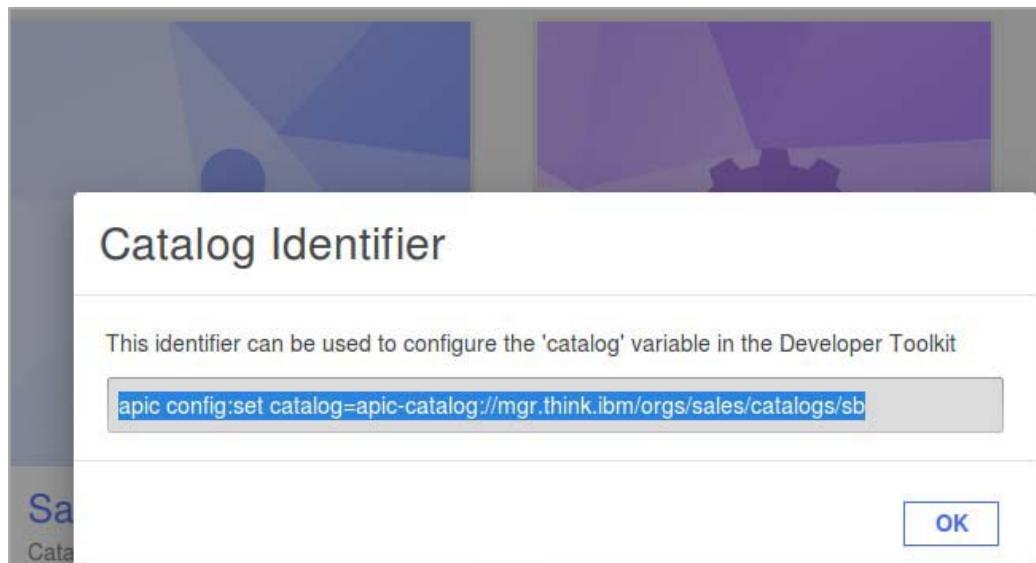
© Copyright IBM Corporation 2017

Figure 13-11. Publish from the terminal: Preparation (1 of 2)



## Publish from the terminal: Preparation (2 of 2)

- Copy the catalog identifier string to a clipboard



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-12. Publish from the terminal: Preparation (2 of 2)

The catalog identifier is required to configure the catalog variable when publishing in the API Developer Toolkit.

Catalog identifier example:

### 1+1=2 Example

apic config:set catalog=apic-catalog://mgr.think.ibm/orgs/sales/catalogs/sb

## Publish a LoopBack project from the terminal

- Sign in to API Manager, if not already signed on

```
apic login -s API_manager_hostname -u username -p password
```

- Paste the command strings for the catalog identifier that you copied earlier in the preparation step into the terminal CLI

```
apic config set catalog=apic-
catalog://API_manager_hostname/orgs/sales/catalogs/sb
```

- Publish the product from the directory of the Loopback project

```
apic publish -s API_manager_hostname definitions/hello-world-
product.yaml
```

The console displays messages that confirm that the product is published

*Figure 13-13. Publish a LoopBack project from the terminal*

1. Sign on to API Manager, unless you are already signed in, with the command:

```
apic login -s API_manager_hostname -u username -p password
where:
```

- API\_manager\_hostname is the server name or IP address of the API Manager
- username is your API Manager organization owner user name
- password is your API Manager password

2. Paste the command strings for the catalog identifier that you copied earlier in the preparation step into the terminal:

```
apic config set catalog=apic-catalog://mgr.think.ibm/orgs/sales/catalogs/sb
```

3. Ensure that your current working directory is the root directory of the LoopBack project (hello-world). Publish the product by typing the command:

```
apic publish -s API_manager_hostname definitions/hello-world-product.yaml
```



## Manage published products in API Manager

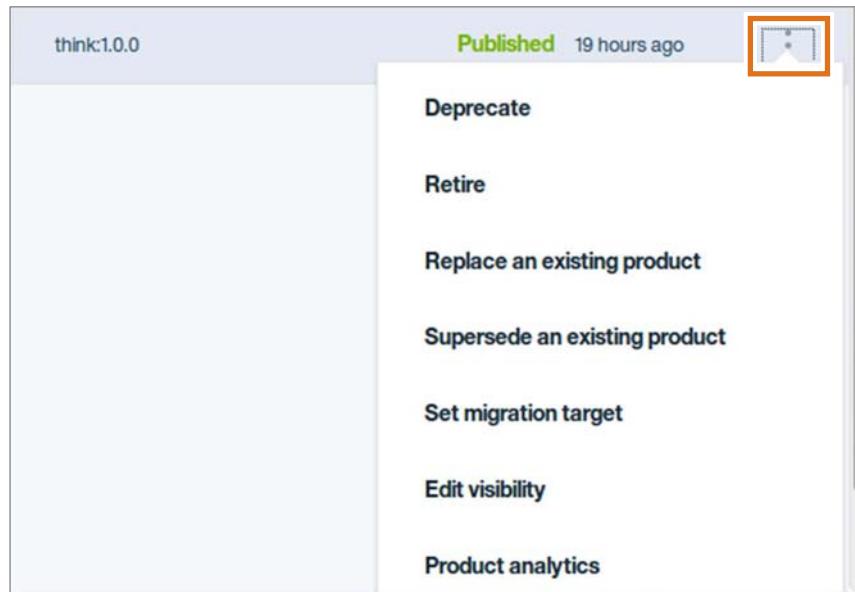
- Published products can be viewed and managed in API Manager

Title	State	Actions
hello-world hello-world:1.0.0	Published	

Figure 13-14. Manage published products in API Manager

## Options on the Manage menu for a published product

- Manage lifecycle and subscription options from the Manage menu



[Staging, publishing, and deploying an API product](#)

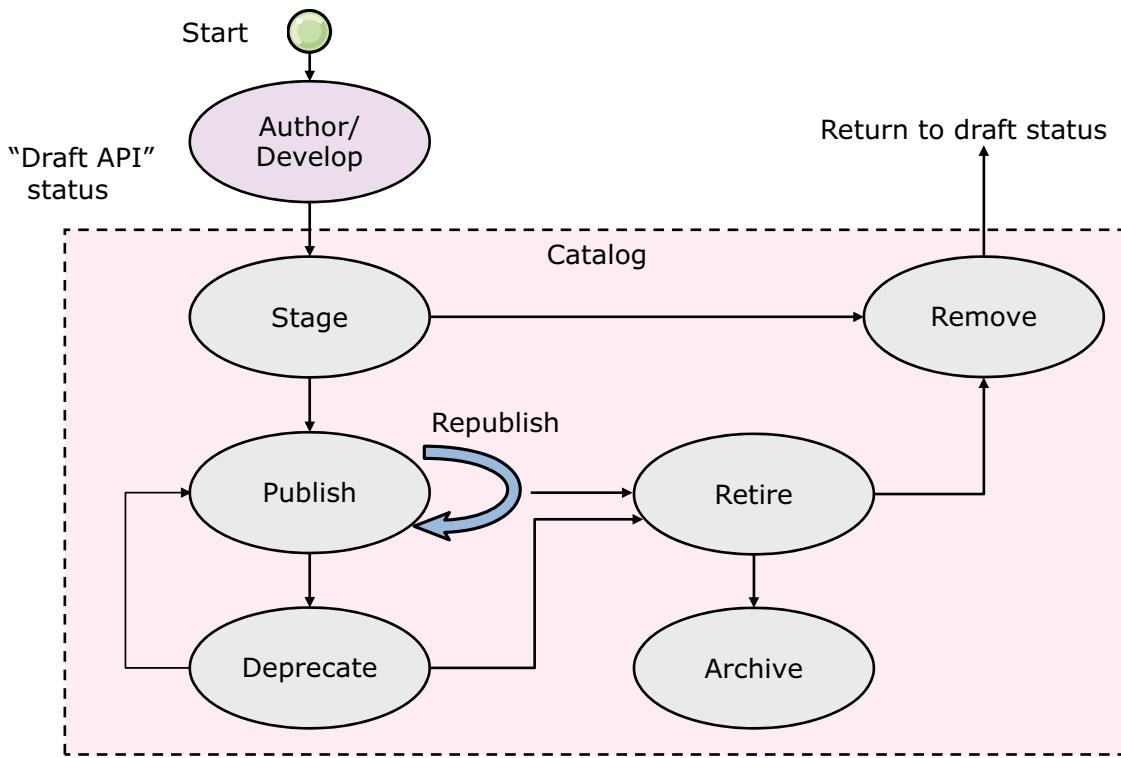
© Copyright IBM Corporation 2017

*Figure 13-15. Options on the Manage menu for a published 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 and viewing product analytics.

## Lifecycle of products and API resources



[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

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 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.

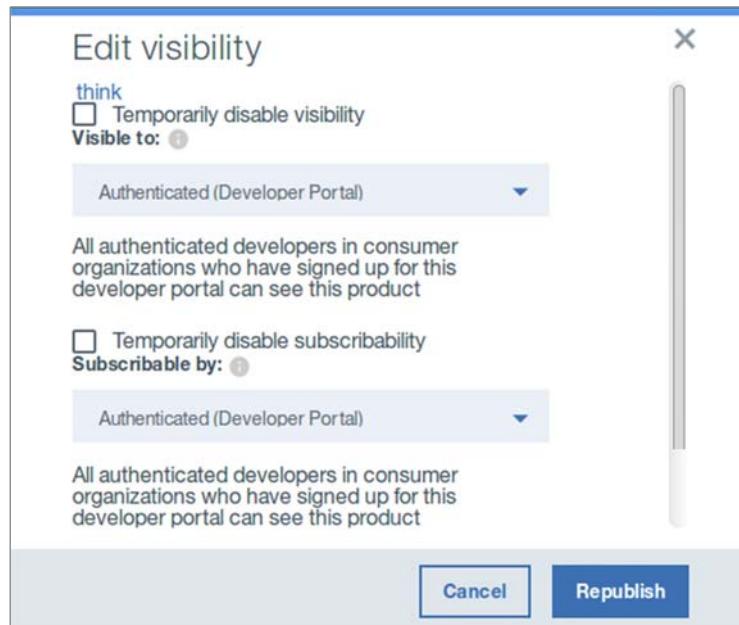
A product version in the retired state cannot be viewed or subscribed to, and all of the associated APIs are stopped.

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.



## Republish a product version

- Change the visibility or subscription options for a previously published product version in API Manager



[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

Figure 13-17. Republish a product version

You can change the visibility and subscription options for a previously published product from the “Edit visibility” window.

Then, you can republish the product.

## Product YAML file: Visibility section

```
visibility:  
  view:  
    enabled: View_enabled  
    type: View_audience  
    orgs: View_organizations  
  subscribe:  
    type: authenticated
```

- *View\_enabled*: true | false
- *View\_audience*: public | authenticated | custom
- *View\_organizations*: organization 1 | organization 2

Figure 13-18. Product YAML file: Visibility section

This page shows the visibility section of the product YAML file that can be set in the “Edit visibility and subscribers” window in API Manager (shown previously).

*View\_enabled* determines whether the product is visible to anybody or not. It must be true or false. If false, the product is not visible in the Developer Portal.

*View\_audience* must be public, in which case the product is visible to anybody who uses the Developer Portal, authenticated, in which case the product is visible to anybody registered through the Developer Portal, or custom, in which case the product is visible to a specified group of users.

*View\_organizations* specifies the organizations that can view the product when *View\_audience* is set to custom.

## Stage and publish a previously published product (1 of 2)

- Target catalog: Sandbox
  - Publishing a product from API Designer to a Sandbox catalog where the product is already published results in the restage and publish of the **same version** of the product
  - Using the test tool or publishing the product overwrites the existing staged and published product even when the APIs are being used in the Developer Portal

[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

*Figure 13-19. Stage and publish a previously published product (1 of 2)*

If you publish a product to a Sandbox catalog and edit it through the products tab of API Designer or API Manager, you can restage and publish the same version of the product.

## 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 Designer
  - Publishing a product from API Designer to a non-sandbox catalog where the product is already published results in another instance of the product
  - For this reason, it is advised that when you stage a product, you then **create a version** of the product in the API Designer to edit in future

[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

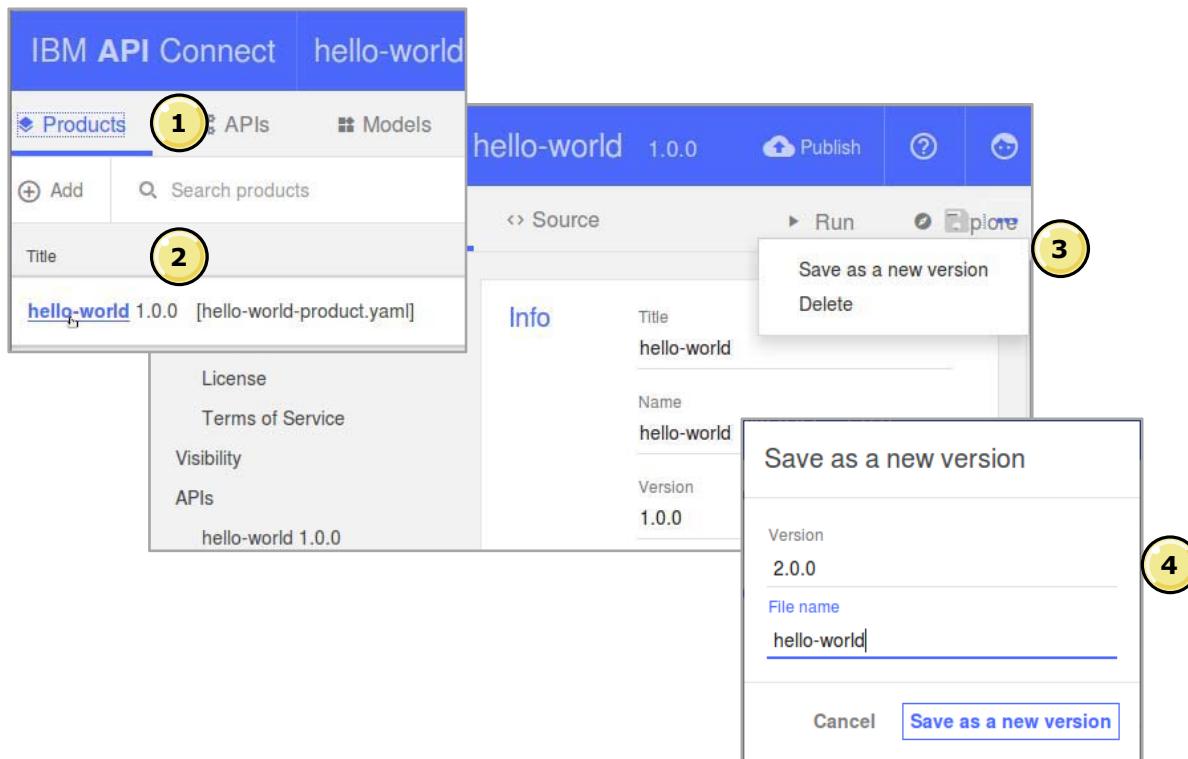
*Figure 13-20. Stage and publish a previously published product (2 of 2)*

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.

When you stage a product, it is suggested that you then create a version of the product to edit in future to avoid confusing the properties of the published product.



## Create a version of a product in the API Designer



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-21. Create a version of a product in the API Designer

In API Designer, you can have multiple versions of a product.

To create a product version:

1. Click the **Products** tab in API Designer. The Products tab opens.
2. Click the product that you want to create your version from. The product details page is displayed.
3. Click the **More Actions** icon. Then, click **Save as a new version**.
4. Type your new version number, and file name. Then, click **Save as a new version**.



## New product version result in the API Designer

The screenshot shows the IBM API Connect interface. At the top, it displays "IBM API Connect" and "hello-world 2.0.0". Below this, there are three tabs: "All Products", "Design" (which is selected), and "Source". On the left, a sidebar titled "Info" lists "Contact", "License", "Terms of Service", "Visibility", "APIs", "Plans", and "Default Plan". On the right, the main panel also has an "Info" section with fields for "Title" (set to "hello-world"), "Name" (set to "hello-world"), "Version" (set to "2.0.0"), and "Description".

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-22. New product version result in the API Designer

The new product version can now be edited in the API Designer.

You can edit the APIs in the product and change them to the same as the product version.



## Permissions for managing catalogs

- From the Settings tab for a catalog, select **Roles**

Role	Catalog Permissions
Administrator Administers the API provider organization	<b>Catalog Settings</b> View and manage the catalog's configuration <input checked="" type="checkbox"/> View <input type="checkbox"/> Manage
Product Manager Manages application developer communities	<b>Catalog Members</b> View and manage catalog's members <input checked="" type="checkbox"/> View <input type="checkbox"/> Manage
API Developer Authors API and product definitions	<b>API Products</b> Stage, view, and manage products in a catalog <input type="checkbox"/> Stage <input checked="" type="checkbox"/> View <input type="checkbox"/> Manage
API Administrator Manages the API product lifecycle	<b>Product Lifecycle Approvals</b> View and approve API product lifecycle state <a href="#">Configure API product lifecycle approval settings</a> <input checked="" type="checkbox"/> View <input type="checkbox"/> Stage <input type="checkbox"/> Publish <input type="checkbox"/> Deprecate <input type="checkbox"/> Retire <input type="checkbox"/> Replace <input type="checkbox"/> Supersede
Catalog Owner Owns and administers the API provider catalog	

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-23. Permissions for managing catalogs

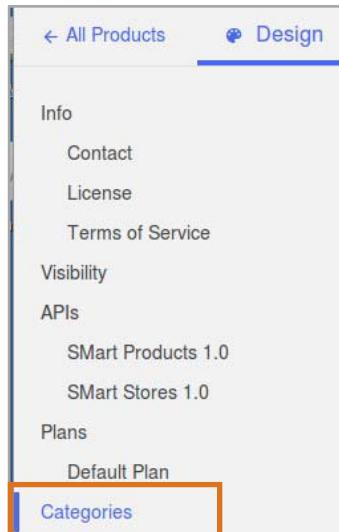
By default, the catalog owner and administrator have all permissions to manage products in sandbox catalogs. Other roles have view permissions by default.

Only the organization owner has permissions to stage, view, and manage products in a non-sandbox catalog.

You can customize the permissions for a catalog from the roles option of the settings tab.

## Product categories (1 of 2)

- You can create a taxonomy within the product by opening the product from the Draft option
  - Select **Categories** in the Design view



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-24. Product categories (1 of 2)

After you created or imported the product definition, click the **Categories** tab from the groupings pane in the Design view.

## Product categories (2 of 2)

- Add the categories to the category list for the product

### Categories

Categories can be added to the product in the form of a hierarchical taxonomy path e.g. 'Animals / Fluffy / Cat'. These values will be reflected in the IBM Developer Portal for a published product. Add each taxonomy path on a separate line with each path element separated by a forward slash.

Category list

```
/Retail  
/Stores  
/Products
```

Figure 13-25. Product categories (2 of 2)

Categories can be added to the product in the form of a hierarchical taxonomy path. These values are reflected in the IBM Developer Portal for a published product. Add each taxonomy path on a separate line with a forward slash separating each path element.



## Publish the product

- Publish the product that contains categories to the catalog

The screenshot shows a user interface for publishing a product. At the top, a green banner displays the message "Success Smart Retail (version 1.0.0) has been published to Sandbox". Below this, the title "Sandbox (Catalog)" is shown with a dropdown arrow, followed by a back arrow labeled "Dashboard". To the right of the title are several icons: a blue diamond, a checkmark, a grid, a double arrow, a bar chart, and a gear. A search bar with the placeholder "Search products" is present. The main content area has two columns: "Title" and "State". A single row is listed: "Smart Retail smart-retail:1.0.0" with the status "Published" in green. The entire interface has a light gray background.

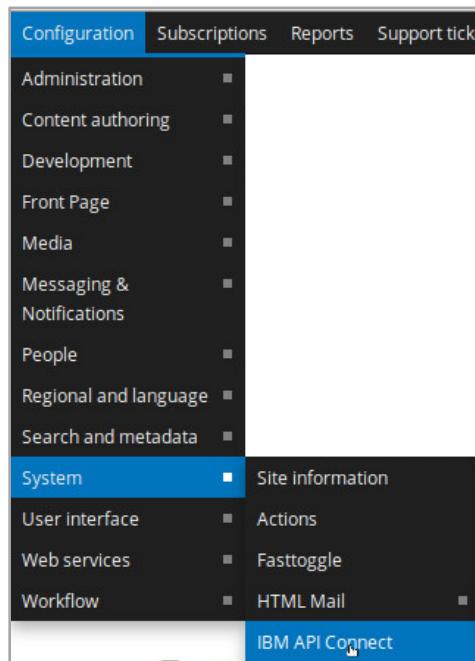
Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-26. Publish the product

## Enable the Developer Portal to display categories (1 of 2)

- Use the administrator dashboard
  - Configuration > System > IBM API Connect



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

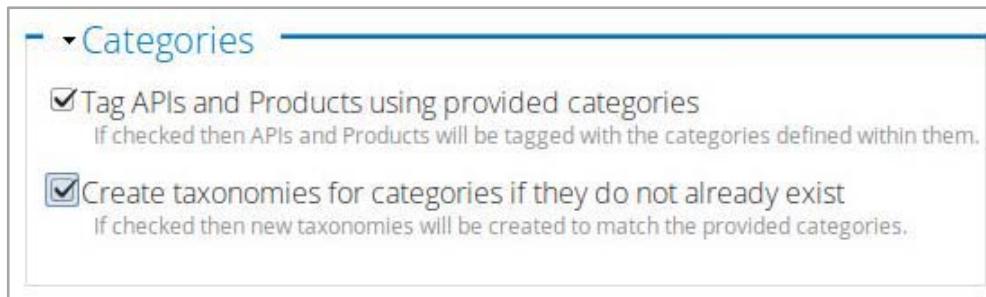
*Figure 13-27. Enable the Developer Portal to display categories (1 of 2)*

Enable the Developer Portal to display the categories for your APIs and products.

On the administrator dashboard, click **Configuration > System > IBM API Connect**.

## Enable the Developer Portal to display categories (2 of 2)

- Select **Create taxonomies for categories**



- Save configuration

Figure 13-28. Enable the Developer Portal to display categories (2 of 2)

In the Categories section, select the **Create taxonomies for categories if they do not already exist** check box.

Click **Save configuration**.



## Categories are displayed in the Developer Portal

A screenshot of the IBM API Connect developer portal. The title bar says "IBM API Connect /dev". Below it, there's a navigation bar with "Home" and "Getting started". On the left, there's a sidebar titled "Smart Retail 1.0.0" under "APIs", with options for "SMart Products" and "SMart Stores". The main content area shows the "View" tab selected for the "Smart Retail 1.0.0" API. It features a purple circular icon with a stack of three squares, the API name "Smart Retail 1.0.0", a five-star rating icon with the text "No votes yet", and a "Taxonomies" section containing "Products, Retail, Stores" which is highlighted with a red border.

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-29. Categories are displayed in the Developer Portal

The categories are displayed as taxonomies in the Developer Portal. When you click one of the categories, the Developer Portal displays the category under the Tags window. You review this pane in the next slide.



## Category tags in the Developer Portal

- The categories are displayed under the Tags heading

A screenshot of the IBM API Connect /dev developer portal. At the top, there's a navigation bar with the IBM logo, the text "IBM API Connect /dev", and links for "Home", "Getting started", and "API Products". Below the navigation bar, the page title is "Tags" and the category name is "Retail". Under "Retail", there's a list of tags: "Retail (1)", "Stores (1)", and "Products (1)". Below the tags, there are three buttons: "View" (underlined), "Edit", and "Translate". To the left of the "View" button is a purple circular icon with a white downward-pointing arrow. To the right of the "View" button is the text "Smart Retail (1.0.0) (2 APIs included)".

IBM API Connect /dev

Home Getting started API Products

Tags

Retail

View Edit Translate

Smart Retail (1.0.0) (2 APIs included)

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-30. Category tags in the Developer Portal

## 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
- Define product categories
- Stage a product to a catalog

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

*Figure 13-31. 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



Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-32. 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: Defining and publishing an API product

Staging, publishing, and deploying an API product

© Copyright IBM Corporation 2017

Figure 13-34. Exercise: Defining and publishing an API product

## Exercise objectives

- Modify the invoke URL for the inventory application to route to the Docker image
- Create a product in the API Designer
- Modify the product properties and add the APIs to the product
- Define an API plan
- Define a publish target
- Stage a product to a catalog



[Staging, publishing, and deploying an API product](#)

© Copyright IBM Corporation 2017

*Figure 13-35. 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 registers for 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
- Explain the difference between self-registration and registration approvals
- Explain how to add an application in the Developer Portal
- Explain the role of client ID and client secret
- Explain how to subscribe to an API plan
- 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 developer 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
- An organization might represent an individual or a group of application developers
- Application developers register their applications and subscribe to plans on the Developer Portal



## 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 an account** link
- Application Developers can create their own user account and organization on the Developer Portal

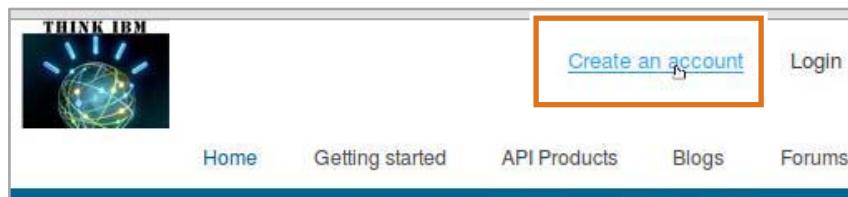


Figure 14-3. Self-registration on the Developer Portal (1 of 5)

Developers can sign up to the Developer Portal without waiting for an approval. Enable the self-service onboarding setting in the API Manager interface.

The Developer Portal then includes a link to create an account.



## Self-registration on the Developer Portal (2 of 5)

- Type the required fields, and then **Create new account**

[Create new account](#)   [Log in](#)   [Request new password](#)

E-mail address \*

A valid e-mail address. All e-mails from the system will be sent to this address. Your e-mail address will also be your username.

**Password Requirements**

- Password must contain characters of at least 3 different types (lowercase, uppercase, digit or punctuation)
- Password must be at least 8 characters in length.

Password \*

 >Password quality: 

Confirm password \*

 Passwords match: yes

Provide a password for the new account in both fields.

First name \*

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-4. Self-registration on the Developer Portal (2 of 5)

The “Create new account” window is shown.

## Self-registration on the Developer Portal (3 of 5)

- User responds by clicking the email link to activate the account in the Developer Portal

Fri, 18 Nov 2016 13:46:16 -0500 (EST)  
From: API Connect Admin <admin@think.com>  
To: newuser@consumer.ibm  
Message-ID: <320064489.31479494755429.JavaMail.root@management.esx.ibm>  
Subject: Thank you for signing up for our APIs  
MIME-Version: 1.0  
Content-Type: text/plain; charset=utf-8  
Content-Transfer-Encoding: 7bit

Hello,

Thank you for signing up for access to APIs from Sandbox.  
To activate your account, click the following link:

[https://developer.think.ibm/sales/sb/?q=ibm\\_apim/activate/x&activationToken=ey](https://developer.think.ibm/sales/sb/?q=ibm_apim/activate/x&activationToken=ey)

## Subscribing and testing APIs

© Copyright IBM Corporation 2017

*Figure 14-5. Self-registration on the Developer Portal (3 of 5)*

The link is displayed in the body of the email message that is sent to the email address that was specified during account creation.



## Self-registration on the Developer Portal (4 of 5)

- User signs on to the Portal with the same credentials that were specified during account creation

The screenshot shows a user login interface. At the top, a green banner displays a checkmark icon and the text "Account successfully activated, please login to continue." Below the banner, the title "User login" is centered. Underneath the title are three navigation links: "Create new account", "Log in" (which is underlined, indicating it is the active link), and "Request new password". The "Log in" section contains two input fields: "Username \*" with the value "newuser@consumer.ibm" and "Password \*" with the value "\*\*\*\*\*". Below each input field is a descriptive placeholder text. A large blue "Log in" button is located at the bottom of the form.

Subscribing and testing APIs

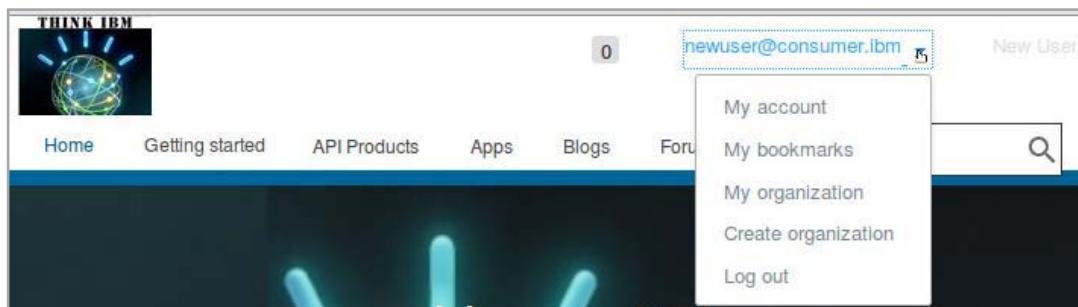
© Copyright IBM Corporation 2017

Figure 14-6. Self-registration on the Developer Portal (4 of 5)



## Self-registration on the Developer Portal (5 of 5)

- User is signed on and can now manage the account



Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-7. Self-registration on the Developer Portal (5 of 5)

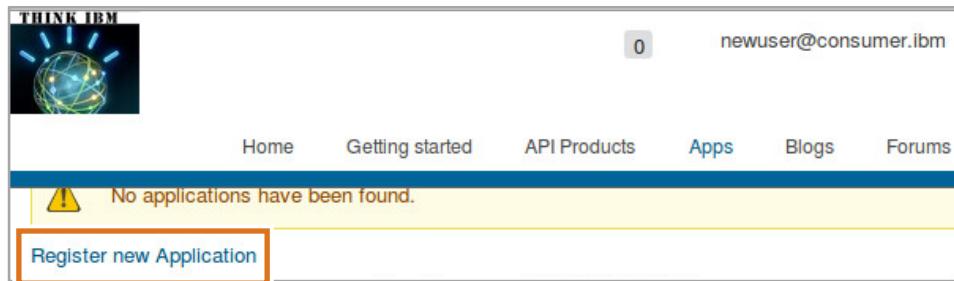
The user is authenticated and signed on to the Developer Portal.

The user can manage the account and organization from the list that is displayed below the user's email address.



## Register an application (1 of 3)

- Select **Apps** from the menu
- Then, click **Register new Application**



Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-8. Register an application (1 of 3)

The link that is named **Register new Application** is available on the **Apps** tab on the Developer Portal.



## Register an application (2 of 3)

- Type the name in the Title field and optional description
- Click **Submit**

Register application

Title \*

Description

OAuth Redirect URI

The URL authenticated OAuth flows for this application should be redirected to.

**Submit**

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-9. Register an application (2 of 3)

In the first wizard window, you specify a title.

Optionally specify a description and an OAuth redirect URI.



## Register an application (3 of 3)

- Type application is created
- Click the **Show** options to display the client secret and client ID

The screenshot shows a successful application creation message: "Application created successfully." and "Your client secret is: qH5qG4cX7sB2JS3iH4eC0gB3wM3oD3IV2vB4hR5gE0dY6dP6xY". A checkbox labeled "Show Client Secret" is checked. Below this, the application details are shown: "Newest Client App" with a circular icon containing a smartphone and gear, last updated on "2016-11-02", and a description "Latest Think App". An "Update" button is present. Under "Client Credentials", the "Client ID" field contains "87fb76fb-bbc5-4383-8850-1ce123e4b886", with a "Show" checkbox checked and a "Reset" button nearby. A "Client Secret" field is also visible.

- Client ID and Client Secret are credentials that an application uses to identify itself

[Subscribing and testing APIs](#)

© Copyright IBM Corporation 2017

Figure 14-10. Register an application (3 of 3)

You can require that, when calling an API operation, 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.



## 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 a user interface for managing subscriptions. At the top, there are fields for 'Client ID' (containing '87fb76fb-bbc5-4383-8850-1ce123e4b886') and 'Client Secret' (a redacted field). To the right of the Client ID field is a checked checkbox labeled 'Show'. Below these fields is a 'Verify' button. The main section is titled 'Subscriptions' and contains the message 'No subscriptions found. Why not browse th' followed by a blue link 'available APIs?'. This link is highlighted with a red rectangle.

Figure 14-11. Subscribe an application to a Product plan (1 of 6)

The **available APIs** link is at the end of the page with the Client ID and Client Secret.

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



Subscribing and testing APIs

© Copyright IBM Corporation 2017

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 are displayed

A screenshot of the IBM API Management interface. At the top, there's a navigation bar with links for Home, Getting started, API Products, Apps, Blogs, and Forums. A user icon shows '1' notification and displays 'newuser@consumer.ibm' with a dropdown arrow. To the right, it says 'New User'. The main content area shows a product named 'think 1.0.0'. On the left, a sidebar lists 'APIs' and several service names: inventory, financing, logistics, and oauth. The main panel shows the 'think 1.0.0' product details, including a blue circular icon with three horizontal bars, a rating of 5 stars, and the message 'No votes yet'. Below that is a 'Description' section with the text: 'The think product will provide really awesome APIs to your application.' There are also sections for 'Contact information' and 'License'.

Subscribing and testing APIs

© Copyright IBM Corporation 2017

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”.



## Subscribe an application to a Product plan (4 of 6)

- Go to the Plans section
- Subscribe to a plan

The screenshot shows the 'Plans' section of the IBM API Management interface. On the left, there's a sidebar with 'think 1.0.0' at the top, followed by a tree view of APIs: 'APIs', 'inventory', 'financing', 'logistics', and 'oauth'. Under each API, there's a list of products. For 'inventory', there's one product: 'inventory 1.0.0' with a 'Gold' plan and 'unlimited' access. For 'financing', 'logistics', and 'oauth', there are two products each: 'financing 1.0.0', 'logistics 1.0.0', and 'oauth 1.0.0', all with 'unlimited' access. At the bottom right of the grid, there's a blue 'Subscribe' button with a white border, which is highlighted with a red rectangle.

API	Product	Plan	Access
inventory	inventory 1.0.0	Gold	unlimited
financing	financing 1.0.0		unlimited
logistics	logistics 1.0.0		unlimited
oauth	oauth 1.0.0		unlimited

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-14. Subscribe an application to a Product plan (4 of 6)

Go to the Plans section of the Product page to view and subscribe to a plan.



## Subscribe an application to a Product plan (5 of 6)

- Select the application that is to be subscribed to the plan
- Click **Subscribe**

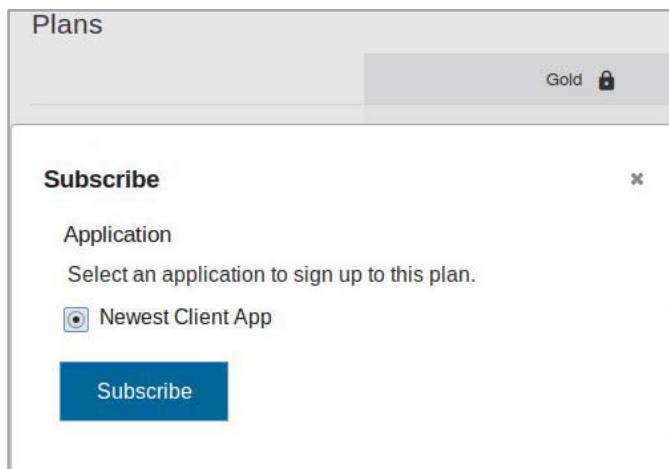
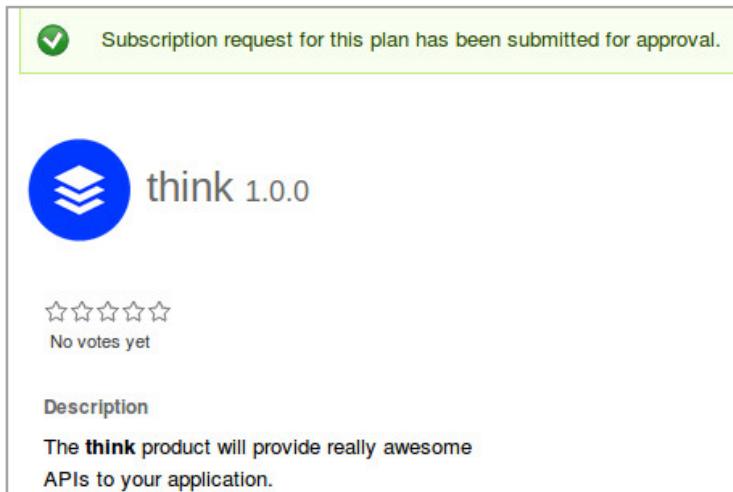


Figure 14-15. Subscribe an application to a Product plan (5 of 6)

Finally, select the application that is being subscribed to the plan.

## Subscribe an application to a Product plan (6 of 6)

- The application is subscribed to the plan
- The application can now call and test the APIs



Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-16. Subscribe an application to a Product plan (6 of 6)

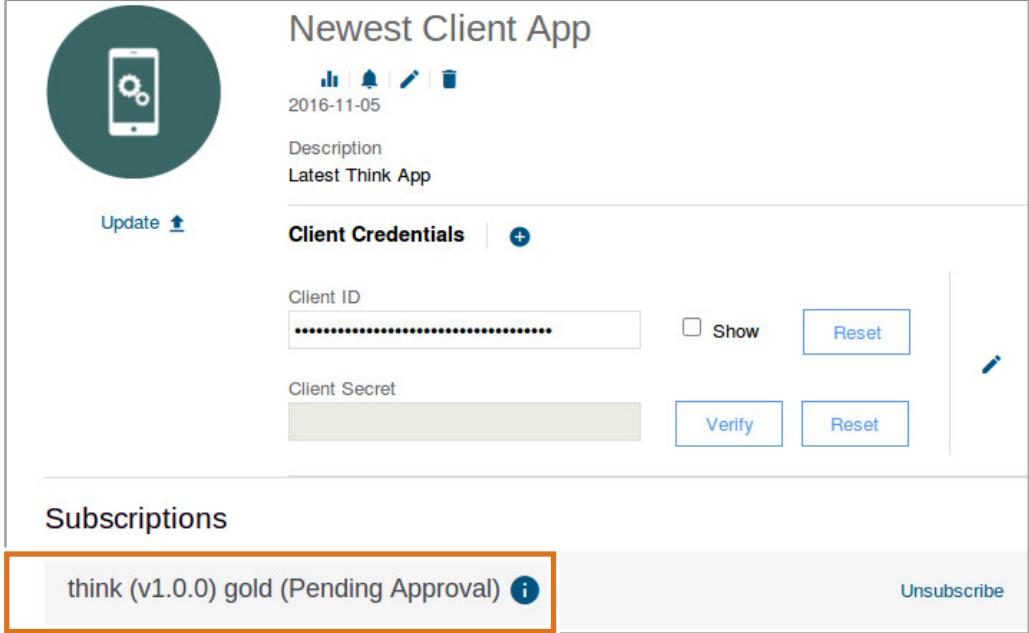
You see a message that reads “Subscription request for this plan has been submitted for approval.”

If automatic subscriptions are enabled for the catalog in API Manager, then the application is automatically subscribed to the plan.

IBM Training 

## Approval requests (1 of 3)

- Verify whether the subscription request is approved



The screenshot shows the 'Newest Client App' page. At the top, there's a circular icon with a smartphone and gears, labeled 'Update ↑'. To its right, the app name 'Newest Client App' is displayed along with icons for metrics, notifications, edit, and delete. Below this, the creation date '2016-11-05' and a brief description 'Latest Think App' are shown. Under 'Client Credentials', there are fields for 'Client ID' (containing a redacted string) and 'Client Secret' (also redacted), each with a 'Show' checkbox and a 'Reset' button. A 'Verify' button is also present. In the 'Subscriptions' section, a single entry 'think (v1.0.0) gold (Pending Approval)' is listed, with an information icon (a blue circle with a white 'i') next to it. This entry is highlighted with an orange border. To the right of the entry is an 'Unsubscribe' button.

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-17. Approval requests (1 of 3)

On this page, you see that the subscription is pending approval.



- Approval list in API Manager
- The owner can approve the request from the **Actions** option

The screenshot shows the IBM API Connect interface with the "Approvals" tab selected. A single approval request is listed:

Requested by	Request to	Submitted	Actions
New User Newest Client App	Subscribe think 1.0.0	3 days ago	<input checked="" type="checkbox"/> <input type="button" value="X"/>

Figure 14-18. Approval requests (2 of 3)

The API producer owner can approve the subscription request in the API Manager user interface.



## Approval requests (3 of 3)

- Verify that the application is subscribed to the plan in the Developer Portal

The screenshot shows the "Newest Client App" page in the IBM Cloud developer portal. At the top, there's a circular icon with a smartphone and gear, followed by the app name "Newest Client App", the creation date "2016-11-05", and a description "Latest Think App". Below this, there's a section for "Client Credentials" with fields for "Client ID" and "Client Secret", each with "Show" and "Reset" buttons. A "Verify" button is also present. In the "Subscriptions" section, a box highlights the entry "think (v1.0.0) gold" with an information icon, and an "Unsubscribe" button is to its right. The entire screenshot is framed by a light gray border.

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-19. Approval requests (3 of 3)

The page shows that the application is subscribed to the gold plan for the think product.

## Testing an API in the Developer Portal

- The API definition must include the statements:

```
x-ibm-configuration:
  testable: true
  enforced: true
  cors:
    enabled: true
```

```
logistics_1.0.0.yaml - Mousepad
File Edit View Text Document Navigation Help
swagger: '2.0'
info:
  x-ibm-name: logistics
  title: logistics
  version: 1.0.0
schemes:
  - https
basePath: /logistics
consumes:
  - application/json
produces:
  - application/json
securityDefinitions:
  clientID:
    description: ''
    in: header
    name: X-IBM-Client-Id
    type: apiKey
security:
  - clientID: []
x-ibm-configuration:
  testable: true
  enforced: true
  cors:
    enabled: true
  gateway: datapower-gateway
Filetype: None Line: 1 Column: 0 OVR ↵
© Copyright IBM Corporation 2017
```

Subscribing and testing APIs

© Copyright IBM Corporation 2017

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 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 7)

- Open a Product from the **API Products** link
- Select the API to be tested

A screenshot of the IBM Developer Portal. At the top, there's a navigation bar with links for Home, Getting started, API Products, Apps, Blogs, and Forums. On the far right, it shows a user icon with '1' and the email 'newuser@consumer.ibm'. Below the navigation, there's a sidebar with links for think 1.0.0, APIs, inventory, financing, logistics, and oauth. The main content area displays the 'think 1.0.0' product page. It features a blue circular icon with three white bars, the text 'think 1.0.0', a five-star rating icon with the text 'No votes yet', and a 'Description' section containing the text: 'The think product will provide really awesome APIs to your application.'.

Subscribing and testing APIs

© Copyright IBM Corporation 2017

*Figure 14-21. Testing an API in the Developer Portal (1 of 7)*

In a typical configuration, an API provider organization uses a development catalog for testing APIs under development. Testing of API operations in a development catalog can be done from the API Designer or from the Developer Portal.



## Testing an API in the Developer Portal (2 of 7)

- Select the operation to be tested

A screenshot of the IBM API Management developer portal. On the left, there's a sidebar with a tree view of APIs: "think 1.0.0", "APIs", "inventory 1.0.0", "financing 1.0.0", and "logistics 1.0.0". The "logistics 1.0.0" item is highlighted with a red box. The main content area shows the "logistics 1.0.0" API details, including a green circular icon with gears, the name "logistics 1.0.0", a "Swagger" download button, and a rating section with five stars and the text "No votes yet". Below this is a "Paths" section with a single entry: "/shipping". Underneath that is a "Definitions" section containing a single item: "GET /shipping".

think 1.0.0

APIs

inventory 1.0.0

financing 1.0.0

logistics 1.0.0

Operations

GET /shipping

GET /stores

Definitions

logistics 1.0.0

No votes yet

Paths

/shipping

GET /shipping

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-22. Testing an API in the Developer Portal (2 of 7)



## Testing an API in the Developer Portal (3 of 7)

- Select the programming language template for the test (default cURL)

The screenshot shows a user interface for testing an API. At the top, there are tabs for different programming languages: cURL (selected), Ruby, Python, PHP, Java, Node, Go, and Swift. A "Subscribe" button is also at the top right. Below the tabs, the section title "Example Request" is displayed. Under this title, a block of cURL command-line code is shown. The code performs a GET request to a URL, specifying headers for accept, content-type, and x-ibm-client-id, and replacing placeholder values with actual ones. Below the code, the section title "Example Response" is visible, indicating where the API's response would be displayed.

```
curl --request GET \
  --url 'https://api.think.ibm/sale
s/sb/logistics/shipping?zip=REPLACE
_THIS_VALUE' \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'x-ibm-client-id: REPLACE
_THIS_KEY'
```

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-23. Testing an API in the Developer Portal (3 of 7)



## Testing an API in the Developer Portal (4 of 7)

- Review the sample response

A screenshot of the IBM Developer Portal interface. At the top, there are tabs for cURL, Ruby, Python, PHP, Java, Node, Go, and Swift, with cURL being the active tab. To the right of the tabs is a "Subscribe" button. Below the tabs, the section title "Example Response" is displayed. Under "Definition", a GET request is shown: "GET https://api.think.ibm/sales/sb/logistics/shipping". Under "Response", a JSON object is displayed:

```
{  
  "xyz": {  
    "next_day": "20.00",  
    "two_day": "17.00",  
    "ground": "8.00"  
  },  
  "cek": {  
    "next_day": "20.00",  
    "two_day": "17.00",  
    "ground": "8.00"  
  }  
}
```

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-24. Testing an API in the Developer Portal (4 of 7)



## Testing an API in the Developer Portal (5 of 7)

- Supply required parameters
- Notice that the **Client ID** is supplied by the application

The screenshot shows the API testing interface in the IBM Developer Portal. At the top, there are tabs for cURL, Ruby, Python, PHP, Java, Node, Go, and Swift, with cURL selected. A "Subscribe" button is also at the top right. Below the tabs, a section titled "Try this operation" contains a URL input field with the value "https://api.think.ibm/sales/sb/logistics/shipping". Under "Identification", there is a "Client ID" dropdown set to "Newest Client App". The "Headers" section includes "content-type: application/json" and "accept: application/json". In the "Parameters" section, there is a "zip" field containing the value "15201".

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-25. Testing an API in the Developer Portal (5 of 7)



## Testing an API in the Developer Portal (6 of 7)

- Click Call operation

The screenshot shows a user interface for testing an API. At the top, there's a "Headers" section with two entries: "content-type" set to "application/json" and "accept" set to "application/json". Below that is a "Parameters" section containing a single entry: "zip" with the value "15201". At the bottom right is a prominent blue button labeled "Call operation".

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-26. Testing an API in the Developer Portal (6 of 7)



## Testing an API in the Developer Portal (7 of 7)

- Review the request and response

The screenshot shows two side-by-side panels. The left panel, titled 'Request', displays a command-line interface (CLI) session. The command entered is 'GET https://api.think.ibm/sales/sb/logistics/shipping?zip=15201'. Below the URL, several environment variables are listed: 'X-IBM-Client-Id: 87fb76fb-bbc5-4383-8850-1ce123e4b886', 'content-type: application/json', and 'accept: application/json'. The right panel, titled 'Response', shows the JSON output of the API call. The response code is '200 OK' and the content type is 'application/json'. The X-Global-Transaction-ID header is present with the value '2089840'. The JSON payload contains two objects: 'xyz' and 'cek'. Both objects have three properties: 'next\_day', 'two\_day', and 'ground'. Their values are: 'xyz' has 'next\_day': '22.03', 'two\_day': '15.07', and 'ground': '11.60'; 'cek' has 'next\_day': '20.03', 'two\_day': '13.70', and 'ground': '10.54'.

```
Request
GET https://api.think.ibm/sales/sb/logistics/shipping?zip=15201
X-IBM-Client-Id: 87fb76fb-bbc5-4383-8850-1ce123e4b886
content-type: application/json
accept: application/json

Response
200 OK
Content-Type: application/json
X-Global-Transaction-ID: 2089840
{
  "xyz": {
    "next_day": "22.03",
    "two_day": "15.07",
    "ground": "11.60"
  },
  "cek": {
    "next_day": "20.03",
    "two_day": "13.70",
    "ground": "10.54"
  }
}
```

Figure 14-27. Testing an API in the Developer Portal (7 of 7)

Examples of the request and response messages are shown on the page.



## Product catalog settings in API Manager (1 of 2)

- Example settings for a non-development catalog

The screenshot shows the 'IBM API Connect' interface with the 'Production' catalog selected. A success message 'Catalog added' is displayed. The 'Settings' tab is active. On the left, a sidebar lists 'Info', 'Gateway', 'Endpoints', 'Portal', 'Permissions', 'Policies', and 'Extensions'. The main panel shows the 'Display Name' set to 'Production' and 'Name' set to 'production'. It includes three toggle options: 'Development mode' (disabled), 'Automatic subscription' (disabled), and 'Default' (disabled). The 'Default' option is described as allowing access via a shorter URL.

Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-28. Product catalog settings in API Manager (1 of 2)



## Product catalog settings in API Manager (2 of 2)

- Set Portal Delegated User Registry

The screenshot shows the 'Settings' tab selected in the top navigation bar. On the left, a sidebar menu under the 'Portal' section includes 'Info', 'Gateway', 'Endpoints', 'Portal' (which is selected), 'Permissions', 'Policies', and 'Extensions'. The main content area displays two configuration sections: 'IBM Developer Portal' and 'Other'. Under 'IBM Developer Portal', the URL is set to <https://developer.think.ibm/sales/production>. Under 'Other', the URL is set to <https://developer.think.ibm/sales/production>. Below these, a section titled 'User Registration and Invitation' contains a dropdown menu labeled 'User Registry' with 'Portal Delegated User Registry' selected. A toggle switch is turned on, followed by the text 'Developers can invite collaborators and assign the following roles:' and a list of three roles: 'Viewer' (description: 'Viewers can only view applications and application activity.'), 'App Developer', and 'API Publisher'.

[Subscribing and testing APIs](#)

© Copyright IBM Corporation 2017

Figure 14-29. Product catalog settings in API Manager (2 of 2)

By enabling the Portal Delegated User Registry in the API Manager UI, you can improve the flexibility of user registry and account management by using the additional configuration options available in the Developer Portal.

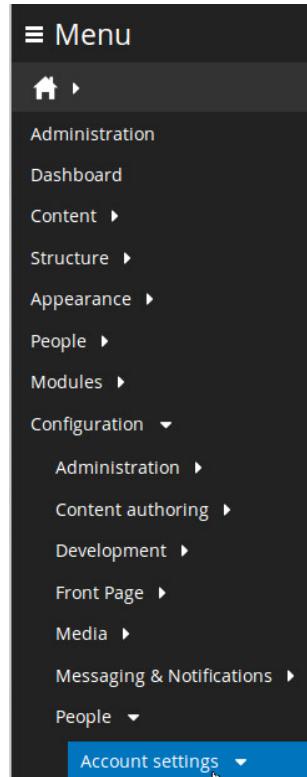
When using the Portal Delegated User Registry, you can specify whether new accounts require administrator approval.

The option Portal Delegated User Registry is required when you want to set up registration approvals. For more information, see the next page.



## Registration approvals (1 of 2)

- From the administration menu, select **Configuration > People > Account Settings**



Subscribing and testing APIs

© Copyright IBM Corporation 2017

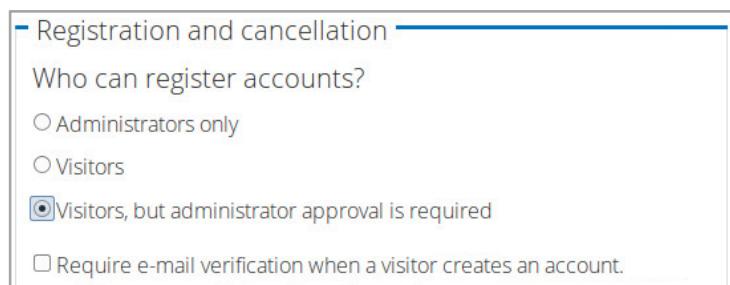
Figure 14-30. Registration approvals (1 of 2)

You must have administrator access to complete this task.

The Portal Delegated User Registry option must be enabled for the catalog in the API Manager UI to complete the task.

## Registration approvals (2 of 2)

- In the **Registration and cancellation** section, select the check box that satisfies your account approval requirements
- Options for approval of new accounts:
  - Administrators only
  - Visitors
  - Visitors, but administrator approval is required



Registration and cancellation

Who can register accounts?

Administrators only

Visitors

Visitors, but administrator approval is required

Require e-mail verification when a visitor creates an account.

Figure 14-31. Registration approvals (2 of 2)

## Unit summary

- Explain the role of application developers in calling published APIs
- Explain the difference between self-registration and registration approvals
- Explain how to add an application in the Developer Portal
- Explain the role of client ID and client secret
- Explain how to subscribe to an API plan
- 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 two of the following options are required to configure approvals for new accounts on the Developer Portal?
  - A. You must have administrator access.
  - B. Self-service onboarding must be disabled.
  - C. Approvals for product lifecycle state changes must be enabled.
  - D. Portal delegated user registry must be selected.



Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-33. Review questions

Write your answers here:

- 1.
- 2.

## Review answers

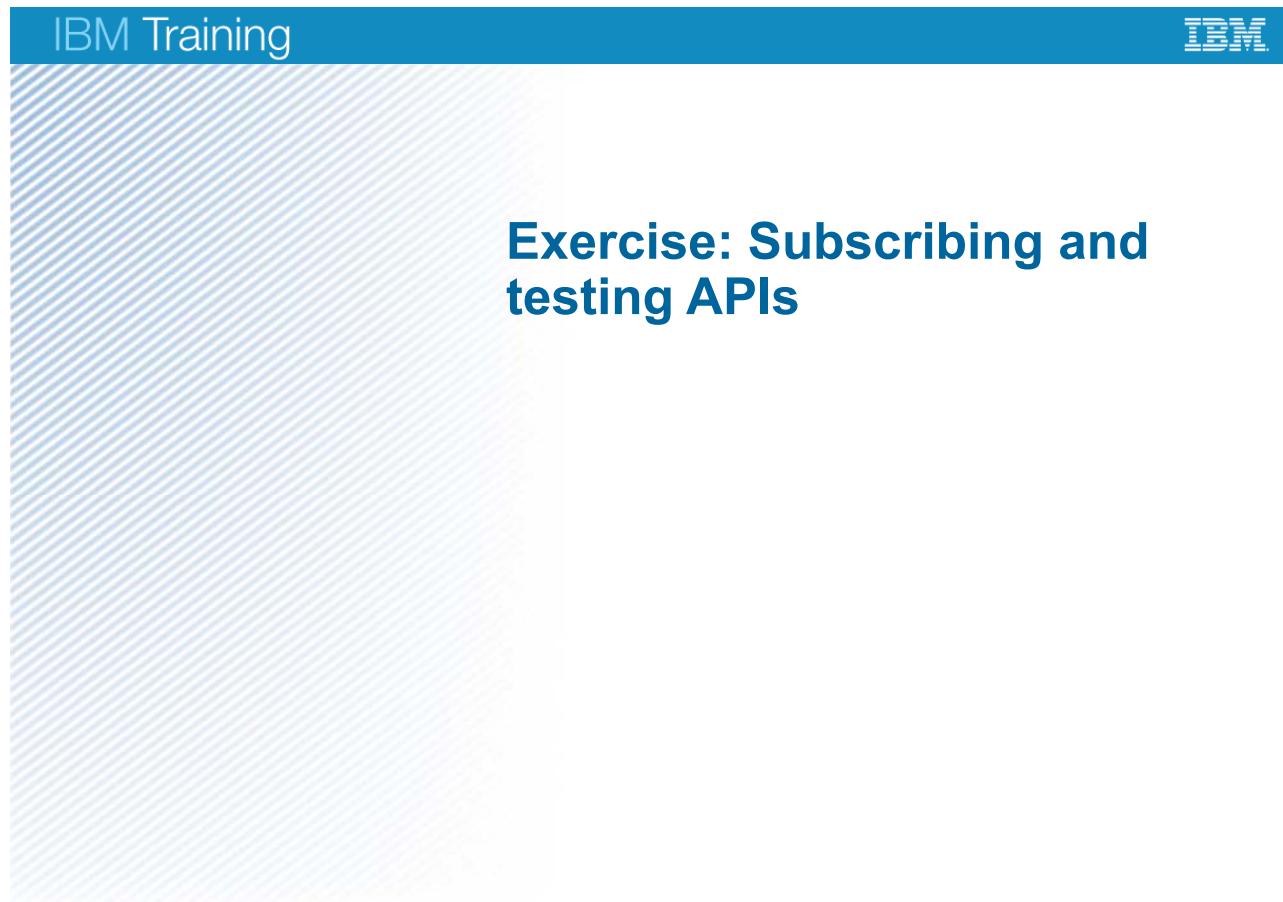
1. True or False: Self-registration can be disabled for the Developer Portal of a development (sandbox) catalog.  
The answer is True.
  
2. Which two of the following options are required to configure approvals for new accounts on the Developer Portal?
  - A. You must have administrator access.
  - B. Self-service onboarding must be disabled.
  - C. Approvals for product lifecycle state changes must be enabled.
  - D. Portal delegated user registry must be selected.The answer is A and D.



Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-34. Review answers



*Figure 14-35. Exercise: Subscribing and testing APIs*

## Exercise objectives

- Self-register an application developer account
- Add an API consumer application in a developer account
- Review and reset the client ID and client secret values
- Test an API operation in the Developer Portal
- Test an API operation with a consumer application



Subscribing and testing APIs

© Copyright IBM Corporation 2017

Figure 14-36. Exercise objectives

# Unit 15. Troubleshooting

## Estimated time

00:30

## Overview

This unit reviews common deployment and runtime issues with IBM API Connect. You learn how to troubleshoot and isolate an error in the API Connect Cloud. You examine the relationship between the API Manager, Gateway server, and Developer Portal at run time. You also learn how to debug issues with an OAuth 2.0 message flow.

## How you will check your progress

- Review questions
- Lab exercise

## Unit objectives

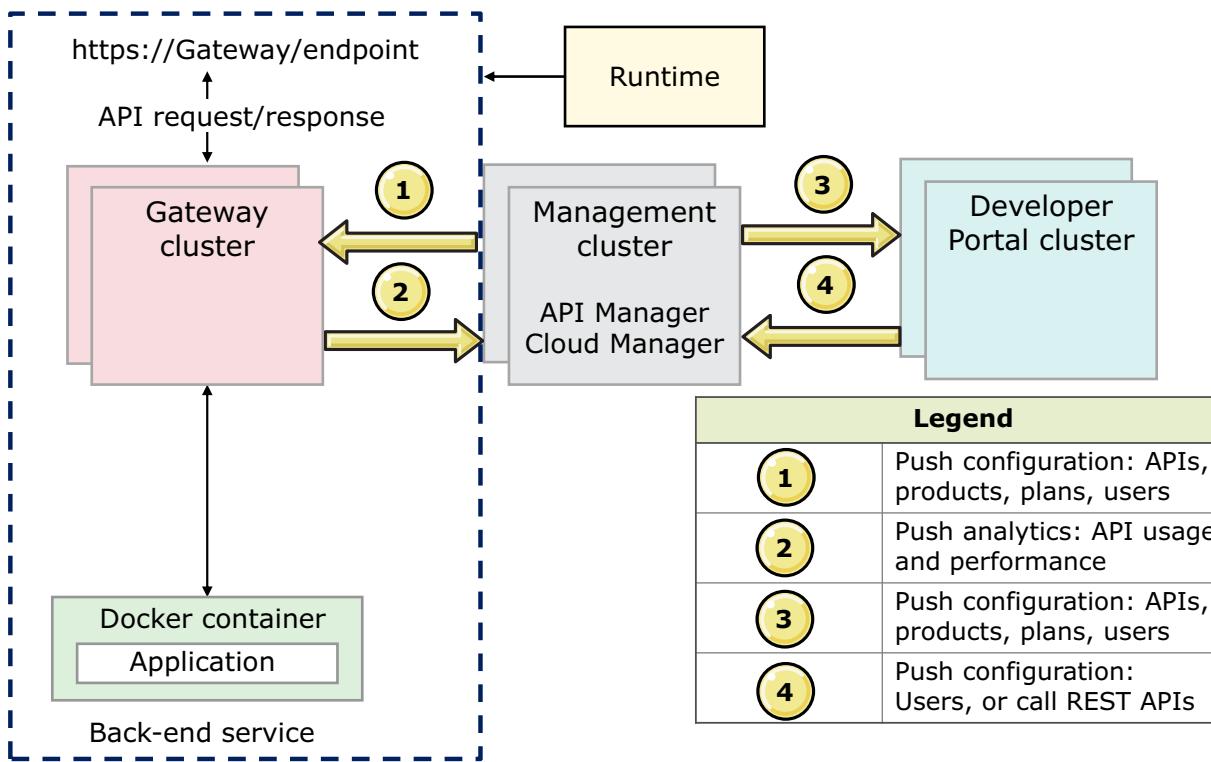
- Identify the API message flow in the API Connect Cloud
- Explain the relationship between the API Manager, Gateway server, and Developer Portal at run time
- Show the proper flow of the consumer web application
- Explain how to troubleshoot errors with an OAuth 2.0 message flow
- Explain how to troubleshoot LoopBack API implementation runtime errors

Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-1. Unit objectives*

## Internal communications at configuration and run time



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-2. Internal communications at configuration and run time

The page shows the internal communications between the Gateway, Management, and Developer Portal servers. The legend shows the message flows that occur mostly at configuration time, when APIs, products, and plans are published. The message flows for analytics occur at run time.

The components and interactions within the dotted area depict the runtime and what occurs when an API endpoint is called on the Gateway server.

The Gateway server handles all the authentication and permissions for access to the products, plans, and APIs that were pushed from the Management server at configuration time.

The Gateway server might call either some external back-end service or a Node.js or Java application that is running on a local runtime or remote Docker container.

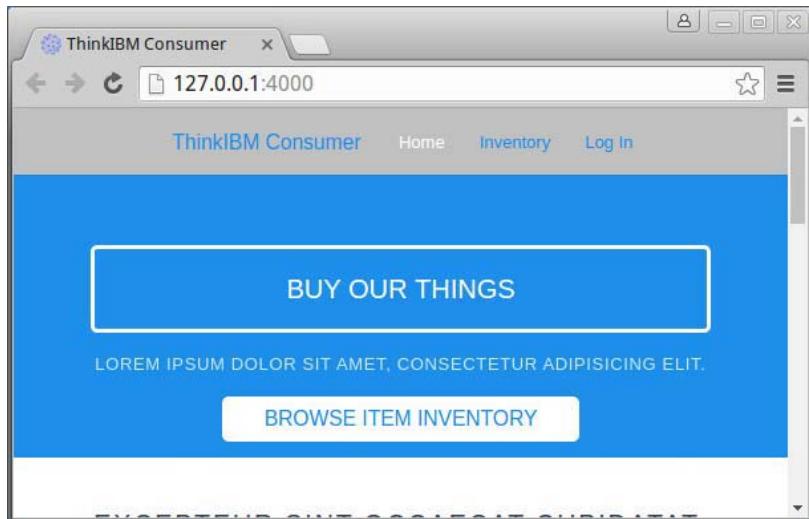
1. Push configuration: Management servers communicate with Gateway servers.
  - Port 443, 5550 HTTPS from Management servers to Gateway servers
2. Push analytics: Gateway servers communicate with Management servers.
  - Port 9443 HTTPS from Gateway servers to Management servers
3. Push configuration: Management servers push configuration to the Developer Portal.
  - Port 22 SSH Management servers to Developer Portal servers
  - Port 443 HTTPS Management servers to Developer Portal servers

4. Pull configuration and make API calls: Developer Portal servers push configuration and call REST APIs.
  - Port 443 HTTPS from Developer Portal servers to Management servers within Management zone
  - Port 2443 from Developer Portal servers to Management servers to subscribe to web hooks



## Running the Consumer web application (1 of 4)

- The ThinkIBM Consumer web client runs as a local Node application
  - Browser opens on 127.0.0.1:4000
  - Click **Log in** or **Browse Item Inventory** to log in



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-3. Running the Consumer web application (1 of 4)

In the exercise case study, the Think IBM consumer web client runs as a local Node application.

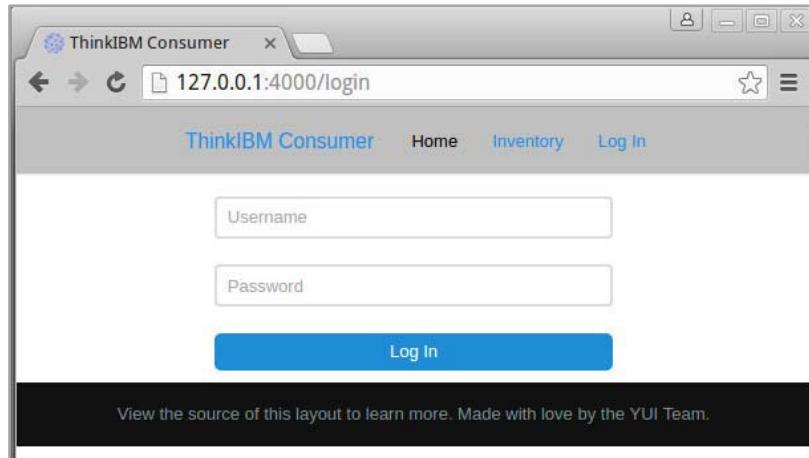
Keep in mind that this application is not the same as a LoopBack Node.js API application.

The browser opens on localhost port 4000.



## Running the Consumer web application (2 of 4)

- ThinkIBM Consumer web client Login page
  - Type any values in the user name and password fields
  - Click **Log In**



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-4. Running the Consumer web application (2 of 4)

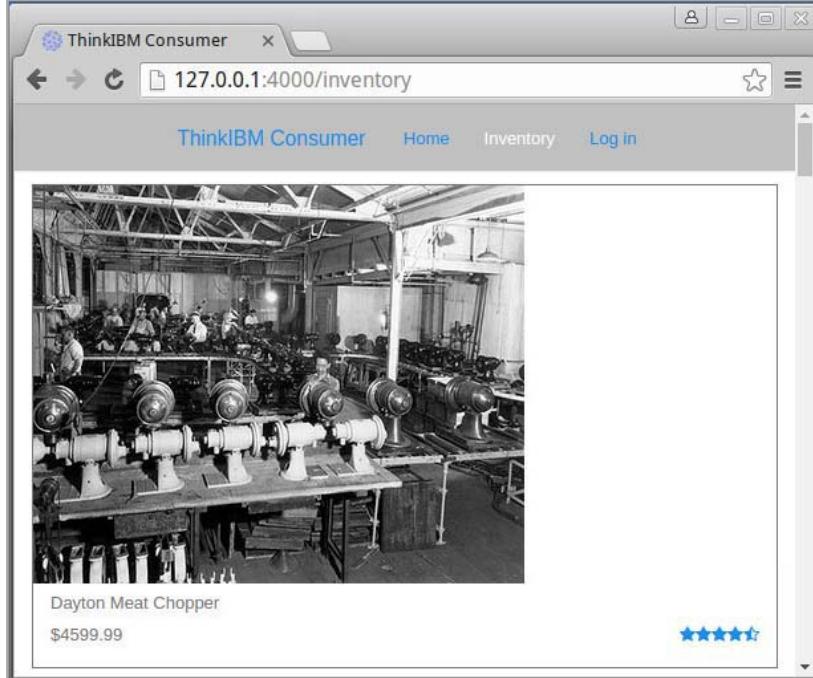
In the exercise case study, the Think IBM consumer web application accepts any value for the user name and password, but you must enter a value. You must not leave these fields blank.

The OAuth application authenticates with any values in these fields.



## Running the Consumer web application (3 of 4)

- Successful calls to all published APIs results in the display of inventory items



Troubleshooting

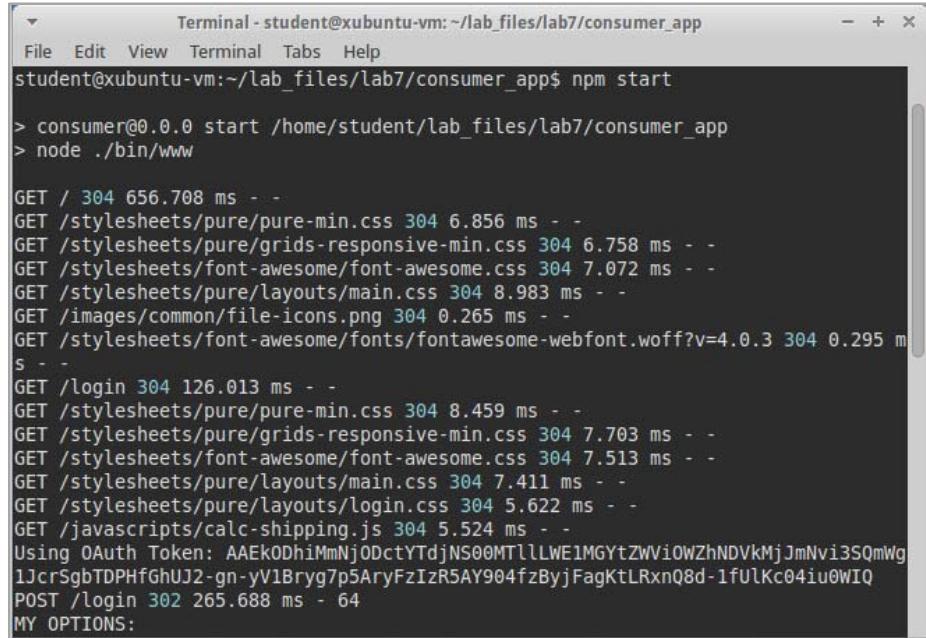
© Copyright IBM Corporation 2017

Figure 15-5. Running the Consumer web application (3 of 4)

If the consumer web application authenticates successfully with the OAuth 2.0 provider API, and it is successful in calling the inventory application that is running in a Docker container, the inventory page is displayed.

## Running the Consumer web application (4 of 4)

- Terminal displays the message Using OAuth Token



```
Terminal - student@xubuntu-vm:~/lab_files/lab7/consumer_app
File Edit View Terminal Tabs Help
student@xubuntu-vm:~/lab_files/lab7/consumer_app$ npm start

> consumer@0.0.0 start /home/student/lab_files/lab7/consumer_app
> node ./bin/www

GET / 304 656.708 ms --
GET /stylesheets/pure/pure-min.css 304 6.856 ms --
GET /stylesheets/pure/grids-responsive-min.css 304 6.758 ms --
GET /stylesheets/font-awesome/font-awesome.css 304 7.072 ms --
GET /stylesheets/pure/layouts/main.css 304 8.983 ms --
GET /images/common/file-icons.png 304 0.265 ms --
GET /stylesheets/font-awesome/fonts/fontawesome-webfont.woff?v=4.0.3 304 0.295 ms --
GET /login 304 126.013 ms --
GET /stylesheets/pure/pure-min.css 304 8.459 ms --
GET /stylesheets/pure/grids-responsive-min.css 304 7.703 ms --
GET /stylesheets/font-awesome/font-awesome.css 304 7.513 ms --
GET /stylesheets/pure/layouts/main.css 304 7.411 ms --
GET /stylesheets/pure/layouts/login.css 304 5.622 ms --
GET /javascripts/calc-shipping.js 304 5.524 ms --
Using OAuth Token: AAEkDhiMmNjODctYTdjNS00MTllWE1MGYtZWVi0WZhNDVkJmNvi3SQmWg
1JcrSgbTDPhfghUJ2-gn-yV1Bryg7p5AryFzIzR5AY904fzByjFagKtLRxnQ8d-1fUlKc04iu0WIQ
POST /login 302 265.688 ms - 64
MY OPTIONS:
```

Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-6. Running the Consumer web application (4 of 4)*

The ThinkIBM Consumer web application terminal output is shown after successfully authenticating with OAuth.

For more information about the details of the console output, see the troubleshooting exercise.

## Calls made by the Consumer application

The Consumer application makes three API calls when you type the user credentials and select Log in:

- Call #1: Consumer app → API Gateway /oauth20/authorize
- Call #2: Consumer app → API Gateway /oauth20/token
- Call #3: Consumer app → API Gateway /inventory/items →  
Docker container with the Inventory LoopBack app

[Troubleshooting](#)

© Copyright IBM Corporation 2017

Figure 15-7. Calls made by the Consumer application

The Consumer application in the lab exercises makes three API calls when you select log in:

- Call #1: Consumer app > API Gateway /oauth20/authorize
- Call #2: Consumer app > API Gateway /oauth20/token
- Call #3: Consumer app > API Gateway /inventory/items >  
Docker container with the Inventory LoopBack app

In *call #1*, the consumer application calls the OAuth 2.0 Provider's authorize API operation.

The authorize operation redirects your web browser to a login page. You enter any value for the user name and password, and the authorization URL returns HTTP status code 200 to the OAuth 2.0 Provider. In turn, the authorize API operation returns an OAuth *authorization bearer token* to the consumer application.

In *call #2*, the consumer application sends the *authorization bearer token* to the token API operation, and a request to the /inventory/items resource. After the token API operation verifies that the bearer token is valid, it returns an OAuth 2.0 *access token* to the consumer application.

In *call #3*, the consumer application calls the /inventory/items API operation with the *access token*.

## Verify that Call #1 and #2 worked

- Check whether the OAuth 2.0 /oauth20/authorize API operation call completed successfully
- If you see the message Using OAuth Token: in your terminal console after you submitted your user name and password, your application completed calls #1 and #2 successfully

Using OAuth Token:

```
AAEkODhiMmNjODctYTdjNS00MT11LWE1MGYtZWViOWZhNDVkJmR64R
66eECxmRzwGsliXpKhZQLBWyEOBiGTL6M2th1gzv1TLzg4eUf5uqjakw
9CixDzNk3Q1twNGUJQqFk3859A
POST /login 302 231.337 ms - 64
```

- If you do not see this message, sign on to the DataPower web user interface
  - Ensure that you are in the API Management domain
  - Select **View Logs** from the Control Panel

Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-8. Verify that Call #1 and #2 worked*

The first two steps in troubleshooting the consumer application are to verify that call #1 and #2 work.

When the console displays the message “Using OAuth Token”, it means that the application completed calls #1 and #2 successfully.

If you do not see that message, check the terminal console for messages. Finally, sign on to the DataPower console and ensure that you are in the API Management domain. Then, select View Logs from the Control Panel.

## Verify that an OAuth token is sent in Call #3

- If call #3 from the consumer app to the /inventory/items API operation fails, you see this error:

```
Inventory API call failed:
```

- Examine the response message in the terminal output

```
Web server listening at: http://localhost:3000
Accessed item matching undefined
Unhandled error for request GET
//inventory/items?filter[order]=name%20ASC: Error: Cannot
GET //inventory/items?filter[order]=name%20ASC
at raiseUrlNotFoundError
(/inventory/node_modules/loopback/server/middleware/url-not-
found.js:21:17)
```

Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-9. Verify that an OAuth token is sent in Call #3

In this specific example, the call to the inventory API failed.

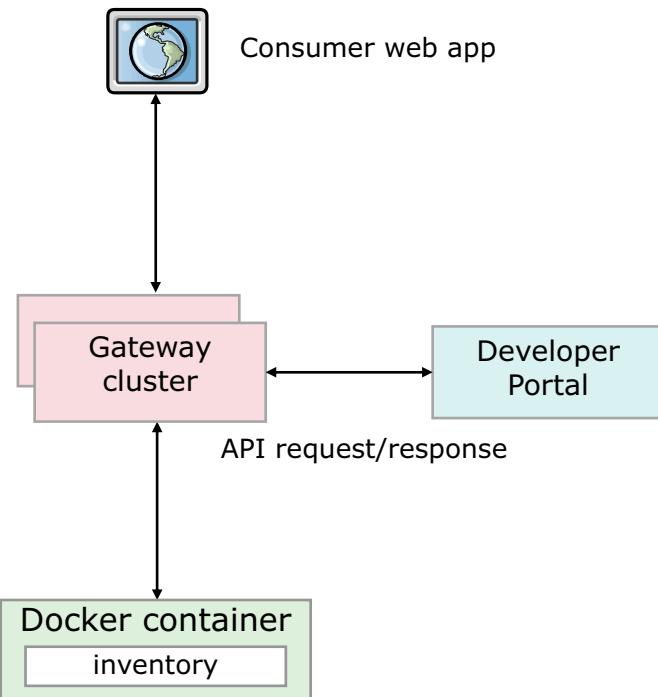
The third call from the consumer API application is to the inventory item's API operation at the gateway. If the operation fails, you see a message in the console log. Next, you must determine whether the condition is an issue with the endpoint target URL of the published API operation, the Loopback API implementation, or the Docker container that runs the LoopBack application. The detailed list of steps on how to troubleshoot these issues is shown next.

## Example: Troubleshooting the inventory application

Steps to troubleshoot the API message:

1. Review the terminal output and the DataPower (gateway) logs for errors
2. Call the API directly on a local Loopback test server
3. Call the API on the Docker container
4. Examine the API properties and target URL
5. Ensure that the API is published
6. Test the API on the Developer Portal
7. Test the API with the consumer web app

[Troubleshooting](#)



[© Copyright IBM Corporation 2017](#)

Figure 15-10. Example: Troubleshooting the inventory application

The page shows the API message flow at run time and the steps that can be taken to troubleshoot runtime errors that occur when running the inventory application.

The inventory application is a Node.js Loopback application that runs on a collective.

The following steps can be taken when the Consumer web application returns errors:

1. Review the terminal output and the DataPower logs for error messages.
2. Test the API on a local Loopback server from the API Designer.
3. Call the API that is running on the Docker container from a browser.
4. Examine the API properties and target URL settings.
5. Verify that the product and API are published.
6. Test the API on the Developer Portal.
7. Test the API with the Consumer application.



## DataPower sign-on

- Sign on to the DataPower Web UI
  - `https://<gateway_IP>:9090`

A screenshot of a web browser displaying the IBM DataPower Gateway sign-on page. The URL in the address bar is `https://dp.think.ibm:9090/login.xml?session=false`. The page title is "IBM DataPower Gateway IDG.7.6.0.0". It displays a message: "IDG console at 192.168.225.52:9090 Your session has expired. Log in again." Below this are four input fields: "User name" containing "admin", "Password" containing "\*\*\*\*\*", "Domain" containing "default", and a dropdown menu for "Graphical Interface" set to "WebGUI". A blue "Login" button is at the bottom.

Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-11. DataPower sign-on

To sign on to the DataPower graphical web interface, type the gateway IP address and port number in the browser URL area. The IBM DataPower Gateway Web UI sign-on page is displayed.



## DataPower logs

- Click **View Logs** from the Control Panel

The screenshot shows the DataPower Gateway Control Panel interface. At the top, it displays the URL "admin @ 192.168.225.52:9090" and links for "Save Configuration" and "Logout". Below this is a "Control Panel" section with a sidebar containing "Control Panel" and "Blueprint Console" options, along with a search bar. The main area is divided into sections: "B2B" (with icons for B2B Partner Profile, B2B Gateway Service, and B2B Transaction Viewer), "Services" (with icons for Web Service Proxy, Multi-Protocol Gateway, and XML Firewall), and "Monitoring and Troubleshooting" (with icons for View Logs, Troubleshooting, and View Status). In the bottom left corner of the main area, there is a link labeled "Troubleshooting". The bottom right corner of the interface contains the copyright notice "© Copyright IBM Corporation 2017".

Figure 15-12. DataPower logs

The View Logs option can be selected from the Control Panel in the DataPower web graphical interface. Errors such as networking errors are displayed in the DataPower logs.

Notice that the Troubleshooting option is also selectable from the Control Panel.



## Call the API directly on localhost

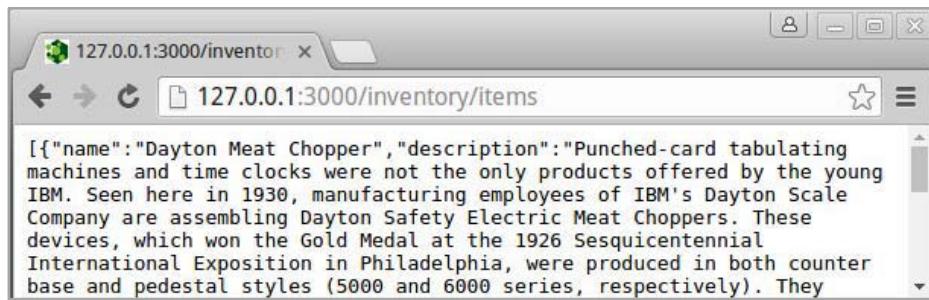
- Start the inventory application from the terminal

```
$ cd ~/inventory
npm start

> inventory@1.0.0 start /home/student/inventory
> node .

Web server listening at: http://0.0.0.0:3000
```

- Test the application locally



Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-13. Call the API directly on localhost*

Start the inventory Node application from the terminal.

Test that the application works on the localhost.

The output is displayed in the browser. In this example, the inventory application is working when tested locally.

Stop the local application with CTRL-C in the terminal.

## Call the API on the Docker container

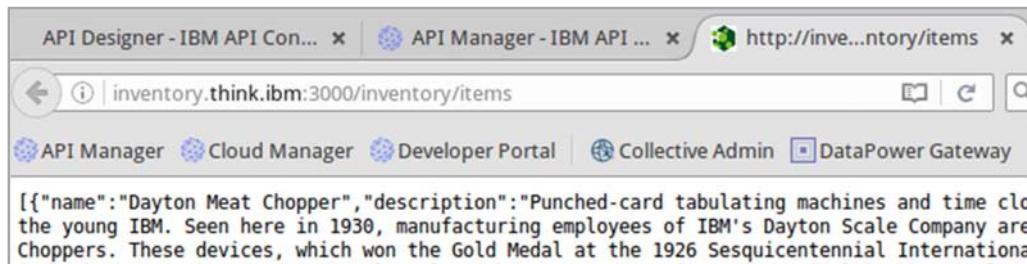
- Start the inventory application on the Docker image

```
$ student@xubuntu-vm:~/inventory$ docker run --add-host
mysql.think.ibm:192.168.225.10 --add-host
mongo.think.ibm:192.168.225.10 -p 3000:3000 apic-inventory-image

> inventory@1.0.0 start /inventory
> node .

Web server listening at: http://localhost:3000
```

- Test the application with the browser



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-14. Call the API on the Docker container

The Loopback inventory application to Docker is deployed independently from the API Manager publishing process.

Ensure that the Docker image is created.

Start the inventory application on the Docker image with the `docker run` command. The container displays a message that the inventory application is started and listening on port 3000 on the container. This port is exposed and mapped to port 3000 on the host by the `docker run` command.

Test the application from a browser session by supplying the URL:

`inventory.think.ibm:3000/inventory/items`

The container should return an array of JSON objects.

Test that the application works on the localhost by using the port mapping to the Docker container.

The output is displayed in the browser. The inventory application is working when it is run on the Docker container.

Leave the Docker container running.

## Examine the API properties

- Examine the API properties in the API Designer or the YAML source

The screenshot shows the 'Design' tab of the API Designer interface. On the left, a sidebar lists various API metadata sections like Consumes, Produces, Lifecycle, etc. Under 'Properties', the 'app-server' property is selected. The main panel displays the property details: 'Property Name \*' is 'app-server' (highlighted), there's an 'Encode' checkbox, and a 'Description' field containing 'Host name and port number'. Below this, a section titled 'Define catalog specific values for this property below' has an 'Add value +' button. A table follows, with columns 'Catalog' and 'Value'. A single row is shown under 'Default' Catalog, where the 'Value' is 'http://inventory.think.ibm:3000' (highlighted).

Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-15. Examine the API properties*

From the API Designer toolkit, examine the properties and values that are defined for the API.

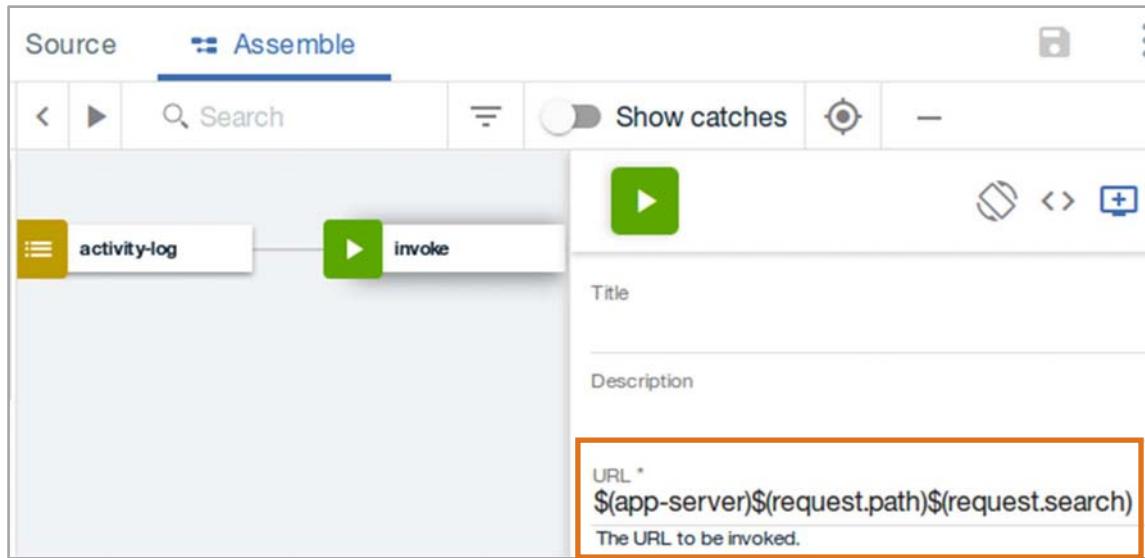
In this example, you examine the default value set for the app-server property that is defined for the inventory API.

Ensure that the property-name and default value matches the name and value that is specified.



## Examine the API target runtime

- Examine the API runtime URL in the API Designer or the YAML source



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-16. Examine the API target runtime

From the API Designer toolkit, examine the values that are set for the runtime URL, edit the API, and then select the **Assemble** tab. Open the properties for the invoke policy.

Examine the properties and values that are defined for the target URL.

Validate the target URL values against the values that are defined in the corresponding exercise.



## Verify that the product and APIs are published

- In API Manager, go to the dashboard for the catalog
- Ensure that the product and APIs are published

The screenshot shows the IBM API Connect dashboard. At the top, there's a navigation bar with "IBM API Connect", "Sandbox", "Explore", a bell icon, a help icon, and a "Sales" dropdown. Below the navigation is a toolbar with icons for "Dashboard", "Catalog", "APIs", "Products", "Subscriptions", and "Metrics". The main area is titled "Dashboard" and contains a table. The table has two columns: "TITLE" and "STATE". It lists several items:

TITLE	STATE
soap-services:1.0.0	Published a month ago
think:1.0.0	Published 11 days ago
financing 1.0.0	Offline / Online
inventory 1.0.0	Online
logistics 1.0.0	Online
oauth-provider 1.0.0	Online
Plans	Subscribers
Silver Plan	1
Gold Plan	0

[Troubleshooting](#)

© Copyright IBM Corporation 2017

Figure 15-17. Verify that the product and APIs are published

Sign on to API Manager.

From the Navigate to icon, open the catalog. Then, review the published products and APIs to ensure that the product and APIs are published.

If you change the API, ensure that the product is republished. When the product is republished, check the product and API status in API Manager.

Also, ensure that an application is subscribed to a product plan.



## Test the API on the Developer Portal

The screenshot shows the IBM Developer Portal interface for testing APIs. On the left, there's a sidebar with language options (cURL, Ruby, Python, PHP, Java, Node, Go, Swift) and a 'Subscribe' button. Below that is a 'Request' section with a pre-filled cURL command:

```
GET https://api.think.ibm/sales/sb/inventory/items
Authorization: Bearer *****
*****
X-IBM-Client-Id: d53ce870-3d74-4c0c-b
46c-0571679b6e2b
X-IBM-Client-Secret: *****
*****
accept: application/json
```

Below the request is a 'Response' section showing a successful 200 OK response:

```
200 OK
Content-Type: application/json; charset=utf-8
X-Global-Transaction-ID: 196c556559f7
```

The main area of the interface is a form for testing the 'inventory' API. It includes fields for Client ID (set to 'Inventory client a'), Client secret, Scope (with 'inventory' checked), Authorization (Username and Password fields), and a Try this operation button.

Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-18. Test the API on the Developer Portal

Sign on to the Developer Portal and select the think product.



### Information

You must sign in to the Developer Portal since only authenticated users can see the think product.

Choose the **inventory** API. Then, select the **GET items** operation.

In the **Try this operation** area, the Client ID drop-down box is prefilled.

You need to paste the **Client secret** into the corresponding field.

In the authorization area, type any non-blank values for the username and password fields.

Click the **Authorize** icon to generate a token.

Finally, click **Call Operation**.

If the call is successful, you see a 200 OK response message.



## Example: Inventory call failed (1 of 4)

- Run the Consumer application
- Sign on to the web application
- Web page displays a call failed message



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-19. Example: Inventory call failed (1 of 4)

Sign on to the consumer web application with a user name and password.

This page is the first of a sequence in troubleshooting the consumer application when an Inventory API call failed message is displayed in the browser.

## Example: Inventory call failed (2 of 4)

- Examine the response message in the terminal output
- Notice that the GET request has a double slash

```
Web server listening at: http://localhost:3000
Accessed item matching undefined
Unhandled error for request GET
//inventory/items?filter[order]=name%20ASC: Error: Cannot
GET //inventory/items?filter[order]=name%20ASC
    at raiseUrlNotFoundError
(/inventory/node_modules/loopback/server/middleware/url-
not-found.js:21:17)
```

Figure 15-20. Example: Inventory call failed (2 of 4)

This page is the second of a sequence in troubleshooting the consumer application when an Inventory API call failed message is displayed in the browser.

Examine the messages in the terminal console window.

In the example, notice that the GET request includes two forward slashes. The console message displays a line that includes the text URLNotFoundError.

## Example: Inventory call failed (3 of 4)

- Review the target URL properties
- Remove the ending slash in the default property value

The screenshot shows the IBM API Designer interface. The top navigation bar has tabs for 'All APIs', 'Design' (which is selected), 'Source', and 'Assemble'. The left sidebar has a tree view with nodes like 'Info', 'Schemes', 'Host', 'Base Path', 'Consumes', 'Produces', 'Lifecycle', 'Policy Assembly', 'Security Definitions', 'Security', 'Extensions', 'Properties' (which is selected), and 'Paths'. Under 'Properties', two items are listed: '/items/{id}/reviews/{fk}' and '/items/{id}/reviews'. The main panel shows the 'app-server' property configuration. It has a 'Property Name \*' field with 'app-server', an 'Encode' checkbox, and a 'Description' field with 'Host name and port number'. Below that, it says 'Define catalog specific values for this property below' and has a 'Add value +' button. A table is shown with columns 'Catalog' and 'Value'. A single row is present with 'Default' in the Catalog column and 'http://inventory.think.ibm:3000/' in the Value column, which is highlighted with a red box.

Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-21. Example: Inventory call failed (3 of 4)

In the example, the default value for the app-server property has an ending slash. Remove the ending slash and save the API.

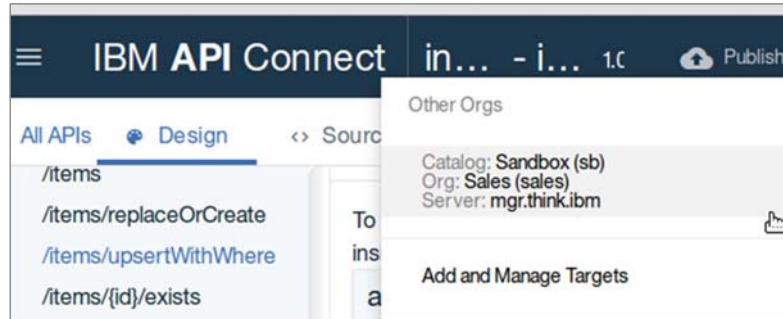
Save the changes.

This page is the third of a sequence in troubleshooting the consumer application when an Inventory API call failed message is displayed in the browser.

Review the properties for the API in the API Designer. The app-server property includes a default URL that ends with a forward slash. Remove the last slash in the URL and save the API.

## Example: Inventory call failed (4 of 4)

- Republish the API



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-22. Example: Inventory call failed (4 of 4)

Publish the product and API to the catalog.

Rerun the consumer application.

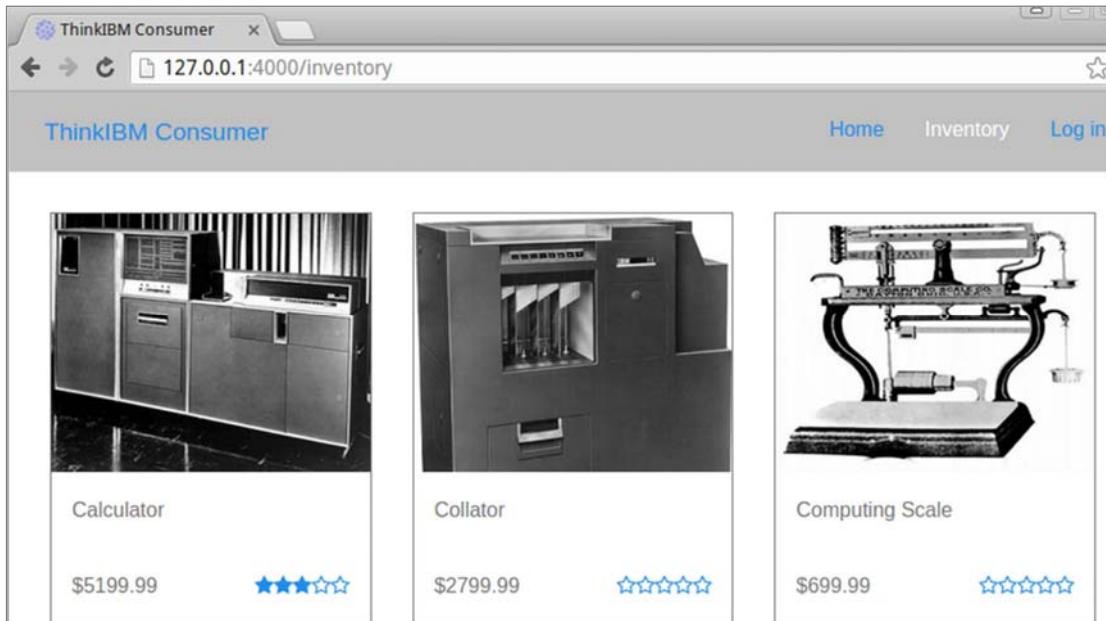
This page is the fourth of a sequence in troubleshooting the consumer application when an Inventory API call failed message is displayed in the browser.

Republish the product and API to the associated catalog.

Then, retest the consumer application.

# IBM Training

## Working Consumer app



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-23. Working Consumer app

The page shows the working consumer application after you type the user ID and password, and after OAuth authenticates the credentials. The page displays images of the different inventory items from the back-end database.

## Unit summary

- Identify the API message flow in the API Connect Cloud
- Explain the relationship between the API Manager, Gateway server, and Developer Portal at run time
- Show the proper flow of the consumer web application
- Explain how to troubleshoot errors with an OAuth 2.0 message flow
- Explain how to troubleshoot LoopBack API implementation runtime errors

Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-24. Unit summary*

## Review questions

1. True or False: The DataPower web graphical user interface has a Troubleshooting pane.
2. True or False: Publishing the product makes the API callable on the gateway and the runtime environment.



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-25. Review questions

Write your answers here:

- 1.
- 2.

## Review answers

1. True or False: The DataPower web graphical user interface has a Troubleshooting pane.  
**The answer is True.**
  
2. True or False: Publishing the product makes the API callable on the gateway and the runtime environment.  
**The answer is False. Publishing makes the API callable only on the gateway. You must deploy the API application independently to the container runtime.**



Troubleshooting

© Copyright IBM Corporation 2017

Figure 15-26. Review answers

## Exercise: Troubleshooting the case study

Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-27. Exercise: Troubleshooting the case study*

## Exercise objectives

- Verify the key components in the exercise case study
- Troubleshoot and correct common issues with the case study
- Isolate implementation issues in your own API Connect Cloud solution



Troubleshooting

© Copyright IBM Corporation 2017

*Figure 15-28. Exercise objectives*

---

# Unit 16. 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 2017

*Figure 16-1. Unit objectives*

## Course objectives

- Create APIs with the API Connect toolkit
- Implement APIs with the LoopBack Node.js framework
- Validate, filter, and transform API requests and responses with message processing policies
- Authorize client API requests with security definitions
- Enforce an OAuth flow with an OAuth 2.0 Provider API
- Stage, publish, and test APIs on the API Connect Cloud

[Course summary](#)

© Copyright IBM Corporation 2017

*Figure 16-2. Course objectives*

## To learn more on the subject

- IBM Training website:  
[www.ibm.com/training](http://www.ibm.com/training)
- API Connect: IBM Developer Center  
<https://developer.ibm.com/apiconnect/>
- API Connect service on IBM Cloud  
<https://console.ng.bluemix.net/catalog/services/api-connect/>
- IBM Knowledge Center for API Connect  
[https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/mapfiles/getting\\_started.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/mapfiles/getting_started.html)
- IBM Redbooks: Getting Started with IBM API Connect: Scenarios Guide  
<http://www.redbooks.ibm.com/abstracts/redp5350.html>

Course summary

© Copyright IBM Corporation 2017

Figure 16-3. 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](http://www.ibm.biz/CloudEduResources)

Course summary

© Copyright IBM Corporation 2017

Figure 16-4. Enhance your learning with IBM resources

## 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 2017

*Figure 16-5. Unit summary*

IBM Training

IBM

## Course completion

You have completed this course:

*Creating, Publishing, and Securing APIs with IBM API Connect  
V5.0.8*



Any questions?

Course summary

© Copyright IBM Corporation 2017

Figure 16-6. Course completion

# 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**

<b>LDAP</b>	Lightweight Directory Access Protocol
<b>LTPA</b>	Lightweight Third Party Authentication

**M**

<b>MB</b>	megabyte
<b>MIME</b>	Multipurpose Internet Mail Extensions
<b>MIT</b>	Massachusetts Institute of Technology
<b>MQ</b>	Message Queue

**N**

<b>NPM</b>	node package manager
------------	----------------------

**O**

<b>OAI</b>	Open API Initiative
<b>OS</b>	operating system

**P****Q****R**

<b>REST</b>	Representational State Transfer
-------------	---------------------------------

**S**

<b>SAML</b>	Security Assertion Markup Language
<b>SDK</b>	software development kit
<b>SHA</b>	secure hash algorithm
<b>SOA</b>	service-oriented architecture
<b>SOAP</b>	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)
<b>SQL</b>	Structured Query Language
<b>SSH</b>	Secure Shell
<b>SSL</b>	Secure Sockets Layer

**T**

<b>TLS</b>	Transport Layer Security
------------	--------------------------

**U**

<b>UI</b>	user interface
<b>UML</b>	Unified Modeling Language
<b>UNIX</b>	Uniplexed Information and Computing System
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>UTF</b>	Unicode Transformation Format

**V**

<b>VM</b>	virtual machine
-----------	-----------------

**W**

<b>WS</b>	web service
<b>WSDL</b>	Web Services Description Language

**X**

<b>XML</b>	Extensible Markup Language
<b>XSLT</b>	Extensible Stylesheet Language Transformation

**Y**

<b>YAML</b>	Yet Another Markup Language
-------------	-----------------------------

**Z**



IBM Training



© Copyright International Business Machines Corporation 2017.