

Course Guide

# Develop, Publish, Manage, and Secure APIs with IBM API Connect V5.0

Course code WU500 ERC 1.0



## May 2016 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 2016.**

**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>xii</b>
<b>Course description .....</b>	<b>xiii</b>
<b>Agenda .....</b>	<b>xiv</b>
<b>Unit 1. Introduction to IBM API Connect V5.0 .....</b>	<b>1-1</b>
Unit objectives .....	1-2
1.1. API Connect Overview .....	1-3
API Connect Overview .....	1-4
Topics .....	1-5
API Connect: Simplified and comprehensive API foundation .....	1-6
What is API Connect? .....	1-7
Create, run, manage, and secure .....	1-8
IBM API Connect: Capabilities delivered .....	1-9
API Connect: Grows with your business needs .....	1-10
Key differences between API Connect offerings .....	1-11
API Connect on-premises .....	1-12
API Connect on Bluemix .....	1-13
IBM API Connect as a dedicated cloud service .....	1-14
1.2. Topology and architecture .....	1-15
Topology and architecture .....	1-16
Topics .....	1-17
API Connect: Component view .....	1-18
API Connect: Topology view .....	1-19
API Connect: Components by role .....	1-20
API Connect: Consumers and providers view .....	1-21
Digital ecosystem and enterprise systems .....	1-22
API runtime message flow .....	1-24
Scenario 1: Implement interaction APIs .....	1-25
Scenario 2: Proxy existing APIs .....	1-26
Manage API runtime environments with Collectives .....	1-27
1.3. Gateway options .....	1-28
Gateway options .....	1-29
Topics .....	1-30
API Gateway options .....	1-31
Gateway features .....	1-32
Gateway capabilities (1 of 2) .....	1-33
Gateway capabilities (2 of 2) .....	1-34
1.4. Software and hardware prerequisites .....	1-35
Software and hardware prerequisites .....	1-36
Topics .....	1-37
API Connect: Requirements .....	1-38
Unit summary .....	1-39
Review questions .....	1-40
Review answers .....	1-41
<b>Unit 2. Create an API with the API Connect toolkit .....</b>	<b>2-1</b>
Unit objectives .....	2-2
API Connect: Topology view .....	2-3

2.1.	Introduction to interaction APIs and LoopBack . . . . .	2-4
	Introduction to interaction APIs and LoopBack . . . . .	2-5
	Topics . . . . .	2-6
	API terminology . . . . .	2-7
	Example: product inventory and price calculator . . . . .	2-8
	Message flows in IBM API Connect . . . . .	2-9
	What is LoopBack? . . . . .	2-10
	Interaction API with a Node.js LoopBack application . . . . .	2-11
2.2.	The IBM API Connect toolkit. . . . .	2-12
	The IBM API Connect toolkit . . . . .	2-13
	Topics . . . . .	2-14
	What is the IBM API Connect toolkit? . . . . .	2-15
	How do you install the toolkit? . . . . .	2-16
	Operating system requirements . . . . .	2-17
	API Connect Toolkit: The apic command . . . . .	2-18
	API Connect Toolkit: The API designer web application . . . . .	2-19
	How to create APIs in API Connect . . . . .	2-20
2.3.	Implement an interaction API with a LoopBack application . . . . .	2-21
	Implement an interaction API with a LoopBack application . . . . .	2-22
	Topics . . . . .	2-23
	Instructor demonstration . . . . .	2-24
	Step 1: Generate a LoopBack application . . . . .	2-25
	Step 2: Define a data source . . . . .	2-26
	Step 3: Create models, properties, and relationships . . . . .	2-27
	Step 4: Implement Interaction API logic . . . . .	2-28
	Step 5: Update the API definition file . . . . .	2-29
	Step 6: Start the API Designer application . . . . .	2-30
	Step 7: Review the API definition file with API Designer . . . . .	2-31
	Step 8: Start the LoopBack application and micro gateway . . . . .	2-32
	Relationship between API gateway and LoopBack . . . . .	2-33
	Step 9: Test the interaction API with the API Explorer . . . . .	2-34
	Example: Test API operations in Explorer . . . . .	2-35
	Unit summary . . . . .	2-36
	Review questions . . . . .	2-37
	Review answers . . . . .	2-38
	Exercise: Introduction to IBM API Connect . . . . .	2-39
	Exercise objectives . . . . .	2-40
<b>Unit 3.</b>	<b>Define an API with the API Designer . . . . .</b>	<b>3-1</b>
	Unit objectives . . . . .	3-2
	User story and role when defining APIs . . . . .	3-3
3.1.	Define the API with Swagger 2.0 . . . . .	3-4
	Define the API with Swagger 2.0 . . . . .	3-5
	Topics . . . . .	3-6
	What is an API definition? . . . . .	3-7
	What is the Swagger OpenAPI specification? . . . . .	3-8
	REST operations . . . . .	3-9
	How API Connect uses the Swagger specification . . . . .	3-10
	Specification format . . . . .	3-11
	File structure . . . . .	3-12
3.2.	Features of the API Designer . . . . .	3-13
	Features of the API Designer . . . . .	3-14
	Topics . . . . .	3-15
	What is the API Designer? . . . . .	3-16
	API Designer Source view . . . . .	3-17
	Specifying an API operation in the API Designer visual view . . . . .	3-18

Reference schema definitions from operations . . . . .	3-19
Schema definitions in the API Designer . . . . .	3-20
Sample schema definition in YAML format . . . . .	3-21
Reference the schema definition in the GET operation . . . . .	3-22
API Designer test feature . . . . .	3-23
Assemble view in API Designer . . . . .	3-24
Assembly extensions in Swagger source . . . . .	3-25
Unit summary . . . . .	3-26
Review questions . . . . .	3-27
Review answers . . . . .	3-28
Instructor demonstration . . . . .	3-29
Instructor demonstration steps . . . . .	3-30
Step 1: Start the API Designer application . . . . .	3-31
Step 2: Define an API . . . . .	3-32
Step 3: Review the API definition . . . . .	3-33
Step 4: Define a path for the API . . . . .	3-34
Step 5: Create the address schema definition for the API . . . . .	3-35
Step 6: Create the branch schema definition for the API . . . . .	3-36
Step 7: Define the GET operation for the API . . . . .	3-37
Step 8: Configure the assembly to call an existing API . . . . .	3-38
Step 9: Review the source for the API definition . . . . .	3-39
Step 10: Start the application . . . . .	3-40
Step 11: Test the API operation from the Assemble tab . . . . .	3-41
Step 12: Select the API operation . . . . .	3-42
Step 13: Enable CORS in the browser (1 of 2) . . . . .	3-43
Step 13: Enable CORS in the browser (1 of 2) . . . . .	3-44
Step 14: Rerun test . . . . .	3-45
<b>Unit 4. Implement an API as a Node.js Loopback application . . . . .</b>	<b>4-1</b>
Unit objectives . . . . .	4-2
API Connect: Topology view . . . . .	4-3
4.1. LoopBack framework in API Connect . . . . .	4-4
LoopBack framework in API Connect . . . . .	4-5
Topics . . . . .	4-6
LoopBack: An open source Node.js API framework . . . . .	4-7
Example: Hello-world LoopBack application . . . . .	4-8
Example: Hello-world note model (1 of 2) . . . . .	4-9
Example: Hello-world note model (2 of 2) . . . . .	4-11
LoopBack framework: models, properties, relationships . . . . .	4-12
LoopBack framework: API mapped to models . . . . .	4-13
LoopBack framework: data persistence with connectors . . . . .	4-14
LoopBack framework: remote methods and hooks . . . . .	4-15
LoopBack framework: non-database connectors . . . . .	4-16
LoopBack framework: packaging . . . . .	4-17
The role of LoopBack in API Connect . . . . .	4-18
LoopBack applications in an API Connect solution . . . . .	4-19
Instructor demonstration . . . . .	4-20
Next steps after implementing the API . . . . .	4-21
4.2. LoopBack models, properties, and relationships . . . . .	4-22
LoopBack models, properties, and relationships . . . . .	4-23
Topics . . . . .	4-24
Steps: LoopBack application, models, properties, relations . . . . .	4-25
Generate a LoopBack application with <i>apic loopback</i> . . . . .	4-26
Example: Create a LoopBack application . . . . .	4-27
LoopBack application: directory structure . . . . .	4-28
Where does LoopBack store model information? . . . . .	4-29

Example: Create a model and properties . . . . .	4-30
LoopBack model inheritance . . . . .	4-31
Example: Model definition file . . . . .	4-32
Example: Model script file . . . . .	4-33
How do you define a relationship between models?	4-34
Example: Create model relations . . . . .	4-35
Example: Relations in the model definition file . . . . .	4-36
Model relation types . . . . .	4-37
<b>4.3. LoopBack data sources and connectors.</b> . . . . .	4-38
LoopBack data sources and connectors . . . . .	4-39
Topics . . . . .	4-40
Steps: LoopBack data sources . . . . .	4-41
What is a LoopBack connector?	4-42
How do You persist model data with connectors?	4-43
Database connectors . . . . .	4-44
Example: Create a memory data source . . . . .	4-45
Example: Data sources configuration file . . . . .	4-46
Example: Model configuration file . . . . .	4-47
Database transactions . . . . .	4-48
Non-database connectors . . . . .	4-50
Unit summary . . . . .	4-51
Review questions . . . . .	4-52
Review answers . . . . .	4-53
Exercise: Create a LoopBack application . . . . .	4-54
Exercise objectives . . . . .	4-55

## **Unit 5. Implement and publish LoopBack API applications** . . . . . 5-1

Unit objectives . . . . .	5-2
API Connect: Topology view . . . . .	5-3
<b>5.1. LoopBack remote methods and hooks.</b> . . . . .	5-4
LoopBack remote methods and hooks . . . . .	5-5
Topics . . . . .	5-6
LoopBack framework: remote methods and hooks . . . . .	5-7
Steps: LoopBack remote methods and hooks . . . . .	5-8
Adding logic to models . . . . .	5-9
What is a remote method?	5-10
Example: Remote method . . . . .	5-11
What is a remote hook?	5-12
Example: Remote hook . . . . .	5-13
What is an operation hook?	5-14
<b>5.2. Node.js application deployment with Collectives</b> . . . . .	5-15
Node.js application deployment with Collectives . . . . .	5-16
Topics . . . . .	5-17
Manage API runtime environments with Collectives . . . . .	5-18
Collectives . . . . .	5-19
Collectives and API Connect . . . . .	5-20
How to deploy an API application to a collective . . . . .	5-21
Step 1: Install and set up a collective controller . . . . .	5-22
Step 2: Install and set up a collective member . . . . .	5-23
Step 3: Add the host to a collective . . . . .	5-24
Step 4: Register the application with API Manager server . . . . .	5-25
Step 5: Publish the application to the API Manager . . . . .	5-26
Unit summary . . . . .	5-27
Review questions . . . . .	5-28
Review answers . . . . .	5-29
Exercise: Customize and deploy a LoopBack application . . . . .	5-30

Exercise objectives .....	5-31
<b>Unit 6. Secure an API with security definitions.....</b>	<b>6-1</b>
Unit objectives .....	6-2
API Connect: Topology view .....	6-3
6.1. API security concepts .....	6-4
API security concepts .....	6-5
Topics .....	6-6
Authentication and authorization: API Security Definitions .....	6-7
How do you secure your APIs in API Connect? .....	6-8
What types of security definitions can you define? .....	6-9
6.2. Identify client applications with API key .....	6-10
Identify client applications with API key .....	6-11
Topics .....	6-12
API key: a unique client application identifier .....	6-13
Enforcing security definitions .....	6-14
Rules for defining client ID and client secret .....	6-15
Example: client ID and client secret in the message header .....	6-16
6.3. Authenticate clients with HTTP basic authentication .....	6-17
Authenticate clients with HTTP basic authentication .....	6-18
Topics .....	6-19
Verifying identity with HTTP basic authentication .....	6-20
Example: storing credentials in HTTP request header .....	6-21
Example: Basic authentication security definition .....	6-22
Message confidentiality with Transport Layer Security .....	6-23
6.4. Introduction to OAuth 2.0 .....	6-24
Introduction to OAuth 2.0 .....	6-25
Topics .....	6-26
What is OAuth? .....	6-27
Example: allow third-party access to shared resources .....	6-28
Example: third-party access to inventory API resources .....	6-29
Before OAuth – sharing user passwords .....	6-30
OAuth Step 1: Resource owner requests access .....	6-31
OAuth Step 2: OAuth client redirection to owner .....	6-32
OAuth Step 3: Authenticate owner with authorization server .....	6-33
OAuth Step 4: Ask resource owner to grant access to resources .....	6-34
OAuth Step 5: Resource owner grants client access to resources .....	6-35
OAuth Step 6: Authorization server sends authorization grant code to client .....	6-36
OAuth Step 7: Client requests access token from authorization server .....	6-37
OAuth Step 8: Authorization server sends authorization token to client .....	6-38
OAuth Step 9: OAuth client sends access token to resource server .....	6-39
OAuth Step 10: Resource server grants access to OAuth client .....	6-40
6.5. Configure OAuth 2.0 security in API Connect. ....	6-41
Configure OAuth 2.0 security in API Connect .....	6-42
Topics .....	6-43
Role of IBM API Connect in the OAuth flow .....	6-44
How to enable OAuth 2.0 authentication .....	6-45
Step 1: Create an OAuth 2.0 provider .....	6-46
Step 2: Configure the OAuth 2.0 provider settings .....	6-47
OAuth 2.0 Provider API: Client types .....	6-48
OAuth 2.0 Provider API: OAuth flow and grant types .....	6-49
Step 2: Authorization and token settings .....	6-50
OAuth 2.0 Provider API: Token settings .....	6-51
Step 3: Create an OAuth 2.0 security definition .....	6-52
Step 3: Create an OAuth 2.0 security definition .....	6-53
Unit summary .....	6-54

Review questions .....	6-55
Review answers .....	6-56
Exercise: Configure and secure an API .....	6-57
Exercise objectives .....	6-58
<b>Unit 7. Assemble an API in the API Designer .....</b>	<b>7-1</b>
Unit objectives .....	7-2
7.1. API policies.....	7-3
API policies .....	7-4
Topics .....	7-5
API policies .....	7-6
Message flows and policies in IBM API Connect .....	7-7
API policy types .....	7-8
Assembly logic and built-in policies .....	7-9
Capabilities of gateway policies .....	7-10
Capabilities of DataPower policies .....	7-11
Adding components to your assembly (1 of 5) .....	7-12
Adding components to your assembly (2 of 5) .....	7-13
Adding components to your assembly (3 of 5) .....	7-14
Adding components to your assembly (4 of 5) .....	7-15
Adding components to your assembly (5 of 5) .....	7-16
Edit components in your assembly .....	7-17
Activity log policy .....	7-18
Invoke policy .....	7-19
Gateway script policy .....	7-20
Gateway mapping policy .....	7-21
7.2. Assemble an API operation for an existing service .....	7-22
Assemble an API operation for an existing service .....	7-23
Topics .....	7-24
Instructor demonstration (1 of 2) .....	7-25
Instructor demonstration (2 of 2) .....	7-26
Step 1: Create an API with Loopback .....	7-27
Step 2: Edit the API .....	7-28
Step 3: Review the draft API .....	7-29
Step 4: Add the schema definition for the API .....	7-30
Step 5: Configure the schema definition properties .....	7-31
Step 6: Define the API paths .....	7-32
Step 7: Specify the API parameters .....	7-33
Step 8: Define responses for the API operations .....	7-34
Step 9: Add an operation-switch in the assembly .....	7-35
Step 10: Configure case 0 in the operation-switch .....	7-36
Step 11: Configure case 1 in the operation-switch .....	7-37
Step 12: Define invoke actions in the assembly .....	7-38
Step 13: Map the case 0 path to an existing API operation .....	7-39
Step 14: Map the case 1 path to an existing API operation .....	7-40
Step 15: Start the application and Micro Gateway .....	7-41
Step 15: Test the API operation with API Explorer .....	7-42
Example: Test an invoke operation in the micro gateway .....	7-43
Unit summary .....	7-44
Review questions .....	7-45
Review answers .....	7-46
Exercise: Advanced API Assembly .....	7-47
Exercise objectives .....	7-48
<b>Unit 8. Publish APIs .....</b>	<b>8-1</b>
Unit objectives .....	8-2

Product, plan, API hierarchy .....	8-3
Define product and plan .....	8-4
Product YAML file .....	8-5
Add a Product .....	8-6
Add an API .....	8-7
Add existing APIs to a Product .....	8-8
Catalogs .....	8-9
Publish a plan and API .....	8-10
Publish a plan and API result .....	8-11
Publish from the terminal: preparation (1 of 2) .....	8-13
Publish from the terminal: preparation (2 of 2) .....	8-14
Publish a LoopBack project from the terminal .....	8-15
Manage published Product in API Manager .....	8-16
Options on the Manage menu for a published Product .....	8-17
Lifecycle of Products and API resources .....	8-18
Republish a Product version .....	8-19
Product YAML file: visibility section .....	8-20
Stage and publish a previously published Product (1 of 2) .....	8-21
Stage and publish a previously published Product (2 of 2) .....	8-22
Create a version of a Product in the API Designer .....	8-23
New Product version result in the API Designer .....	8-24
Permissions for managing catalogs .....	8-25
Unit summary .....	8-26
Review questions .....	8-27
Review answers .....	8-28
Exercise: Working with API Products .....	8-29
Exercise objectives .....	8-30
<b>Unit 9. Use an API through the Developer Portal .....</b>	<b>9-1</b>
Unit objectives .....	9-2
User stories and roles when using APIs .....	9-3
9.1. Browse and sign on to the Developer Portal to explore APIs .....	9-4
Browse and sign on to the Developer Portal to explore APIs .....	9-5
Topics .....	9-6
Accessing the Developer Portal .....	9-7
View published products in API Manager .....	9-8
Developer Portal: public interface .....	9-9
Create an account on the Developer Portal (1 of 2) .....	9-10
Create an account on the Developer Portal (2 of 2) .....	9-11
Sign in to the Developer Portal .....	9-12
Adding a user to a Developer Organization .....	9-13
More APIs are visible to authenticated users .....	9-14
9.2. Create an application that uses APIs .....	9-15
Create an application that uses APIs .....	9-16
Topics .....	9-17
Register an application (1 of 2) .....	9-18
Register an application (2 of 2) .....	9-19
Subscribe the application to a plan (1 of 6) .....	9-20
Subscribe the application to a plan (2 of 6) .....	9-21
Subscribe the application to a plan (3 of 6) .....	9-22
Subscribe the application to a plan (4 of 6) .....	9-23
Subscribe the application to a plan (5 of 6) .....	9-24
Subscribe the application to a plan (6 of 6) .....	9-25
9.3. Test an API on the Developer Portal .....	9-26
Test an API on the Developer Portal .....	9-27
Topics .....	9-28

APIs that can be tested on the Developer Portal .....	9-29
Test the API on the Developer Portal (1 of 4) .....	9-30
Test the API on the Developer Portal (2 of 4) .....	9-31
Test the API on the Developer Portal (3 of 4) .....	9-32
Test the API on the Developer Portal (4 of 4) .....	9-33
<b>9.4. Customize the Developer Portal .....</b>	<b>9-34</b>
Customize the Developer Portal .....	9-35
User story and role when customizing the Developer Portal .....	9-36
Topics .....	9-37
Customize the Developer Portal .....	9-38
The administrative menu in the Developer Portal .....	9-39
Install a custom theme (1 of 5) .....	9-40
Install a custom theme (2 of 5) .....	9-41
Install a custom theme (3 of 5) .....	9-42
Install a custom theme (4 of 5) .....	9-43
Install a custom theme (5 of 5) .....	9-44
Customize the Welcome Banner (1 of 3) .....	9-45
Customize the Welcome Banner (2 of 3) .....	9-46
Customize the Welcome Banner (3 of 3) .....	9-47
Developer Portal with the new Welcome Banner and theme .....	9-48
Unit summary .....	9-49
Review questions .....	9-50
Review answers .....	9-51
Exercise: Consumer Experience .....	9-52
Exercise objectives .....	9-53
<b>Unit 10. Exercise case study .....</b>	<b>10-1</b>
Unit objectives .....	10-2
Lab 1: Hello World .....	10-3
Lab 2: Create the Inventory API (1 of 2) .....	10-4
Lab 2: Create the Inventory API (2 of 2) .....	10-5
Lab 3: Customize and publish the Inventory API (1 of 2) .....	10-6
Lab 3: Customize and publish the Inventory API (2 of 2) .....	10-7
Lab 4: Secure the Inventory API (1 of 3) .....	10-8
Lab 4: Secure the Inventory API (2 of 3) .....	10-9
Lab 4: Secure the Inventory API (3 of 3) .....	10-10
Lab 5: Advanced API assembly (1 of 2) .....	10-11
Lab 5: Advanced API assembly (2 of 2) .....	10-12
Lab 6: Publishing an API Product .....	10-13
Lab 7: Customize the Developer Portal .....	10-14
Consumer Application (1 of 10) .....	10-15
Consumer Application (2 of 10) .....	10-16
Consumer Application (3 of 10) .....	10-17
Consumer Application (4 of 10) .....	10-18
Consumer Application (5 of 10) .....	10-19
Consumer Application (6 of 10) .....	10-20
Consumer Application (7 of 10) .....	10-21
Consumer Application (8 of 10) .....	10-22
Consumer Application (9 of 10) .....	10-23
Consumer Application (10 of 10) .....	10-24
Unit summary .....	10-25
<b>Unit 11. Course summary .....</b>	<b>11-1</b>
Unit objectives .....	11-2
Course objectives .....	11-3
Course objectives .....	11-4

Enhance your learning with IBM resources .....	11-5
Course completion .....	11-6

---

# 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®  
SPSS®

Cognitive Era™  
Watson Avatar®

Cognos®  
Worklight®

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

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

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

# Course description

## Develop, Publish, Manage, and Secure APIs with IBM API Connect V5.0

**Duration: 3.5 days**

### Purpose

In this course, you learn how to build a Loopback application with the IBM API Connect toolkit.

You define an API interface that conforms to the Swagger 2.0 OpenAPI specification.

You assemble an API implementation in the API Designer web interface. You stage, publish, and secure APIs in an API gateway.

This course also teaches application developers how to register API applications in the Developer Portal.

The course teaches you how to customize the home page of the Developer Portal.

### Audience

This course is intended for API developers.

### Prerequisites

Before taking this course, students should have a general knowledge of application programming interfaces, Representational State Transfer (REST) architecture, and Node.js programming. To acquire this knowledge, students can attend the following self-paced course:

- VY102 ERC 3.0: Developing REST APIs with Node.js for IBM Bluemix.

### Objectives

- Explain the role of IBM API Connect V5.0 in a digital ecosystem
- Identify the API creation, run, manage, and secure features in IBM API Connect V5
- Create an API with the API Connect toolkit
- Define an API interface with the API Designer
- Call an existing system API
- Implement an API as a Node.js LoopBack application
- Secure an API with OAuth 2.0 authorization
- Assemble API message flows in API Designer
- Publish API and Products with API Management Server
- Create an application on the Developer Portal that uses an API

---

# Agenda

---



## Note

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

---

## Day 1

- (00:30) Course introduction
- (01:15) Unit 1. Introduction to IBM API Connect V5.0
- (01:00) Unit 2. Create an API with the API Connect toolkit
- (01:00) Exercise 1. Introduction to IBM API Connect
- (01:15) Unit 3. Define an API with the API Designer

## Day 2

- (01:15) Unit 4. Implement an API as a Node.js Loopback application
- (01:30) Exercise 2. Create a LoopBack application
- (01:00) Unit 5. Implement and publish LoopBack API applications
- (00:45) Exercise 3. Customize and deploy an application
- (01:30) Unit 6. Secure an API with security definitions

## Day 3

- (01:15) Exercise 4. Configure and secure an API
- (01:15) Unit 7. Assemble an API in the API Designer
- (01:15) Exercise 5. Advanced API assembly
- (01:00) Unit 8. Publish APIs
- (01:00) Exercise 6. Working with API Products

## Day 4

- (01:00) Unit 9. Use an API through the Developer Portal
- (01:15) Exercise 7. API consumer experience
- (00:30) Unit 10. Exercise case study
- (00:10) Unit 11. Course summary

---

# Unit 1. Introduction to IBM API Connect V5.0

## Estimated time

01:00

## Overview

In this unit, you learn about the purpose and the scope of the IBM API Connect V5.0 solution. You examine the role of API Connect in a system of engagement architecture. You examine the API creation, run, secure, and manage features. You identify the software and hardware prerequisites for API Connect.

## Unit objectives

- Explain the digital ecosystem and enterprise core architectures
- Identify the components in the IBM API Connect V5.0 solution
- Identify the API creation, run, manage, and secure features
- Identify the software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

*Figure 1-1. Unit objectives*

## 1.1. API Connect Overview

## API Connect Overview

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

*Figure 1-2. API Connect Overview*

## Topics

### API Connect Overview

- Topology and architecture
- Gateway options
- Software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-3. Topics

## API Connect: Simplified and comprehensive API foundation



Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-4. 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. From API creation, service runtime administration, API security, to lifecycle management.

## 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 and Java support for creating API implementations
  - Integrated enterprise grade clustering, management, and security for Node.js and Java
  - 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

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

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 ecosystem for digital applications: mobile, web, internet of things, partner applications, and internal applications.

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

## Create, run, manage, and secure

Enterprise focused	Developer focused
<p><b>Comprehensive API solution</b> End-to-end integrated experience across API lifecycle – create, run, manage, secure, socialize, and analyze APIs through single offering on-premises, in the cloud or hybrid.</p>	<p><b>Create and run APIs</b> Rapidly create APIs, connect to data sources, and expose them as REST APIs with a model-driven approach. Run Node.js and Java runtimes with unified operations and management.</p>
<p><b>Built-in assembly user experience and policies</b> Use a visual tool to compose API policy flows and built-in policies to secure, control, and optimize API traffic without writing custom code or logging on to the gateway</p> <p><b>Intuitive interface</b> Modern user experience to reduce complexity, improve performance, and allow quicker creation, management, and enforcement of APIs.</p>	<p><b>First class developer experience</b> Enable developers to create and test APIs locally on their workstations in minutes and stage it to on-premises or cloud environments.</p> <p><b>Developer toolkit</b> Enable automated scripting and DevOps automation through a command-line environment for defining, managing, and deploying APIs.</p>

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-6. Create, run, manage, and secure

On the left side, the enterprise focused features 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. The API Connect Developer Toolkit allows developers to start building, testing, and publishing APIs from their own laptops.

## IBM API Connect: Capabilities delivered

### Create

- Rapid model-driven API creation
- Data source to API mapping automation
- Standards-based visual API spec creation in Swagger 2.0
- Local API creation and testing
- On-cloud and on-premises staging of APIs, Plans, and Products

### Secure

- Policy enforcement
- Enterprise security
- Quota management and rate limiting
- Content-based routing
- Response caching, load-balancing, and offload processing
- Message format and transport protocol mediation

### Run

- Node.js and Java API compute instances
- Node.js and Java integrated runtime management
- Enterprise high availability and scaling
- On-cloud and on-premises staging of API applications

### Manage

- API discovery
- API, Plan, and Product policy creation
- API, Plan, and Product Lifecycle Management
- Self-service, customizable, developer portal
- Advanced analytics
- Subscription and community management

[Introduction to IBM API Connect V5.0](#)

© Copyright IBM Corporation 2016

*Figure 1-7. IBM API Connect: Capabilities delivered*

The features in IBM API Connect V5.0 are divided into four themes: API creation, API runtime services, API security, and API management.

## API Connect: Grows with your business needs



Deploy where it is most convenient for you:

- IBM Bluemix
- Third-party cloud provider
- On-premises

Introduction to IBM API Connect V5.0

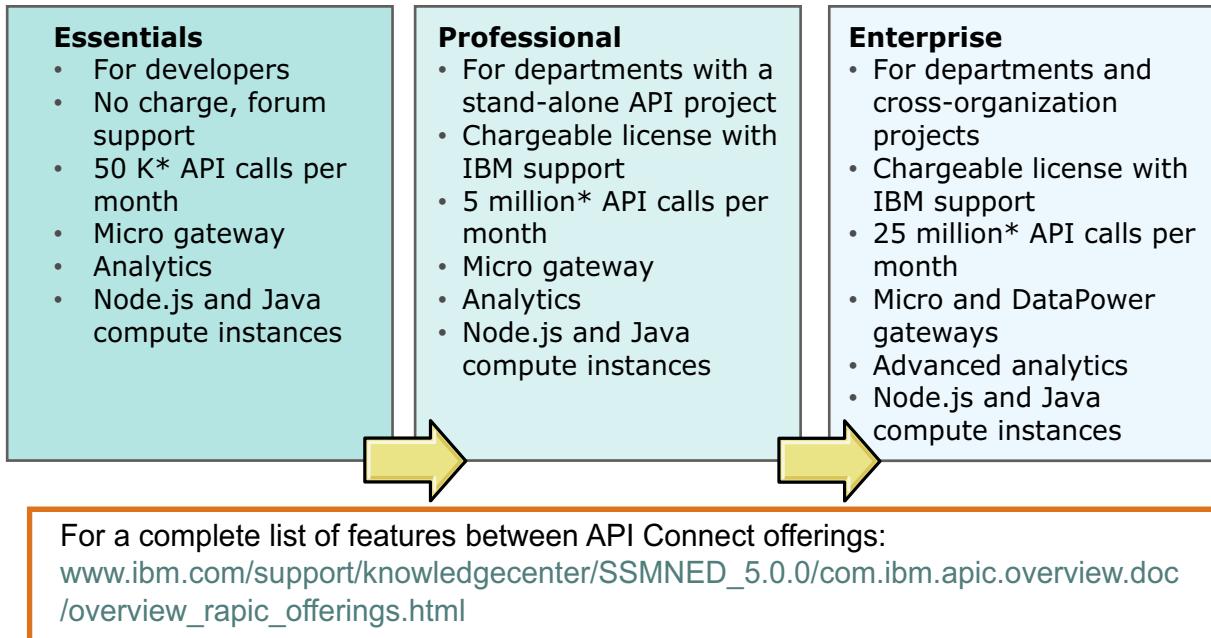
© Copyright IBM Corporation 2016

Figure 1-8. API Connect: Grows with your business needs

There are three offerings for the IBM API Connect V5.0 product: enterprise, professional, and essentials. You learn about the capabilities of each offering on a later slide.

## Key differences between API Connect offerings

- Consumption-based subscription options for customer-managed and IBM managed (Bluemix) offerings.



Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-9. Key differences between API Connect offerings

\*Refer to the website for the most up-to-date API call entitlements for each offering level.

For more information on the feature list in each offering, see:

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

## API Connect on-premises

- Optionally install IBM API Connect on your own infrastructure.
  - Deploy the Developer Portal, Management Server, and Cloud Manager as virtual machines.
  - Install and configure a virtual or physical DataPower appliance as the API Gateway.
  - Alternatively, install the micro gateway as a Node package.
  - Deploy your Node.js or Java API implementations on a Compute runtime.
  - Install and configure a Collective Controller to manage API implementations and micro gateways.

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-10. API Connect on-premises

A collective is a group of multiple servers from a single management domain. The API Connect collective controller manages member servers in the collective. Each collective member hosts a micro service, a server runtime environment for API implementations, or both. The server runtime environment for API implementations is known as a compute runtime.

## API Connect on Bluemix

- **Publish** your API definitions and services right from notebook to the cloud
- **Manage** any service: internal, external, enterprise-owned, or third-party
  - Define, design, import, and discover APIs
  - Apply security, transformation, rate-limit policies
  - Manage APIs, Products, users, subscribers
  - Share and socialize APIs on integrated developer portal
  - Analyze runtime usage to gain visibility and insight
- **Enforce** service consumption policies
  - Enforce subscription and API access at design time
  - Enforce security and control policies at run time
- **Share APIs** with your own developers or business partner developers in Bluemix
- **Identical capabilities** of on-premises, except for infrastructure administration and direct gateway configuration access
- A resilient and **highly available API runtime** infrastructure with built-in failover, redundancy, and dynamic scaling on **IBM SoftLayer**



Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-11. API Connect on Bluemix

The IBM Bluemix platform hosts two components of the IBM API Connect solution: the Developer Portal and the API Management server. With the Bluemix offerings, you subscribe to API Connect in the same manner as any Bluemix service. Bluemix manages the Portal and Management server infrastructure on your behalf. The API Connect Bluemix service runs on top of the IBM SoftLayer platform.

## IBM API Connect as a dedicated cloud service

The power and simplicity of API Connect – offered on the lines of Bluemix Dedicated in your own dedicated SoftLayer environment to both the public Bluemix and your own network.



### Dedicated to you

Single tenant hardware that is dedicated to you – to satisfy regulatory and legal compliance.

### Feels like home

Secure, fast, and unmetered access. APIC Dedicated sits on your network by way of VPN or direct network connectivity.

### Focus on apps, no iron

Focus on building the APIs to succeed in API Economy. IBM manages the platform.

### Global – so you are local

Get closer to your users. APIC Dedicated can live in any of 20+ SoftLayer data centers around the world.

### Pay smart

Never over buy again. Pay based on capacity and adjust capacity monthly depending on needs.

### On call 24 x 7

Experts are always on call to solve problems. Premium support options available at request.

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-12. IBM API Connect as a dedicated cloud service

The dedicated cloud service is an IBM API Connect platform on SoftLayer hardware that is dedicated to you. You have the option to integrate your corporate LDAP server for authentication. In contrast to the Bluemix offering, you have access to the gateway configuration. You can develop and deploy custom policies for your APIs.

## 1.2. Topology and architecture

## Topology and architecture

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

*Figure 1-13. Topology and architecture*

## Topics

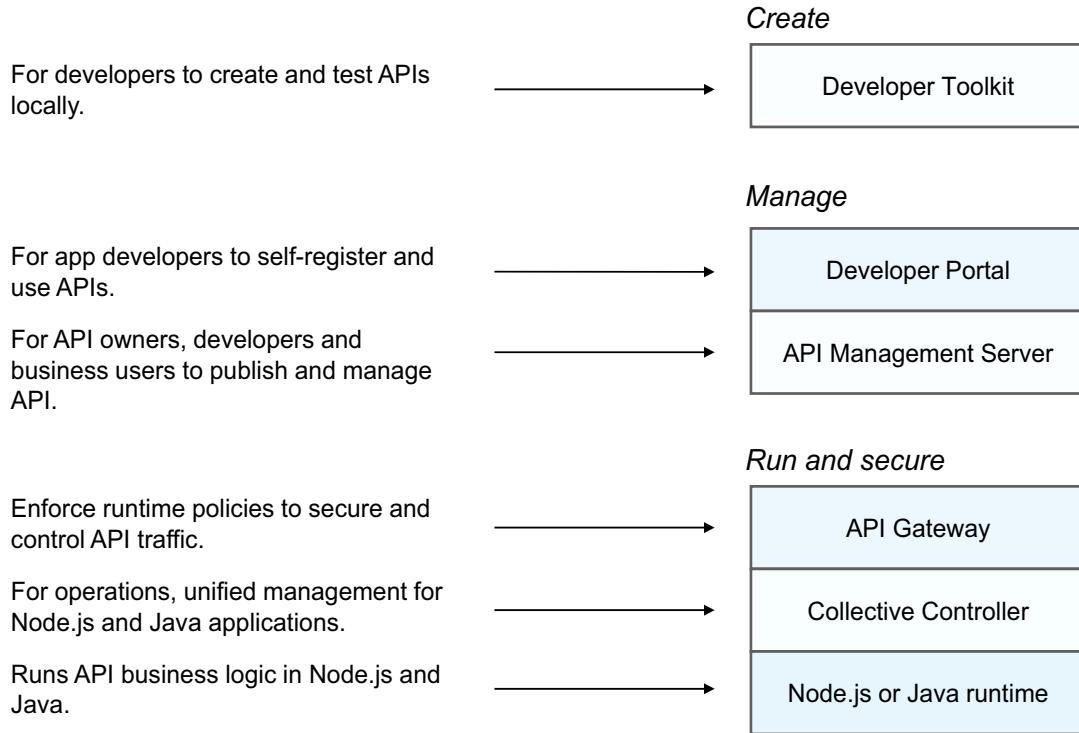
- API Connect Overview
- ▶ Topology and architecture
- Gateway options
- Software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-14. Topics

## API Connect: Component view



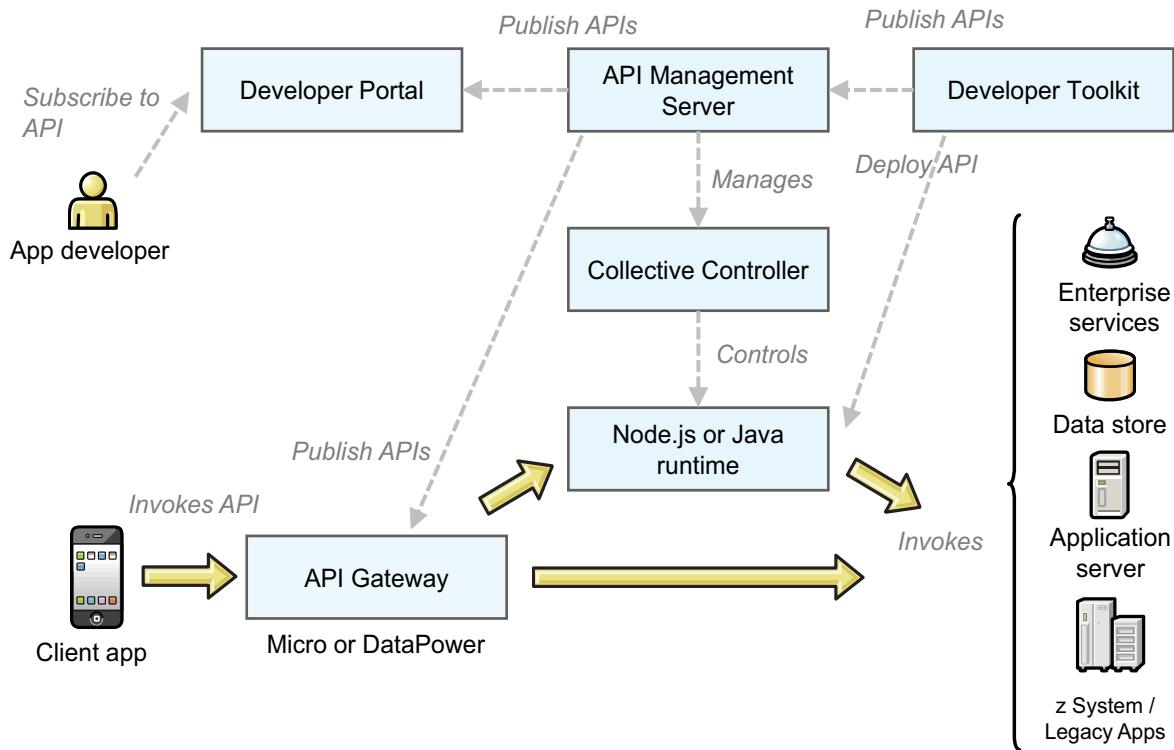
Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-15. API Connect: Component view

This slide identifies the six components that make up IBM API Connect. The list of the right explains the role of each component.

## API Connect: Topology view



Introduction to IBM API Connect V5.0

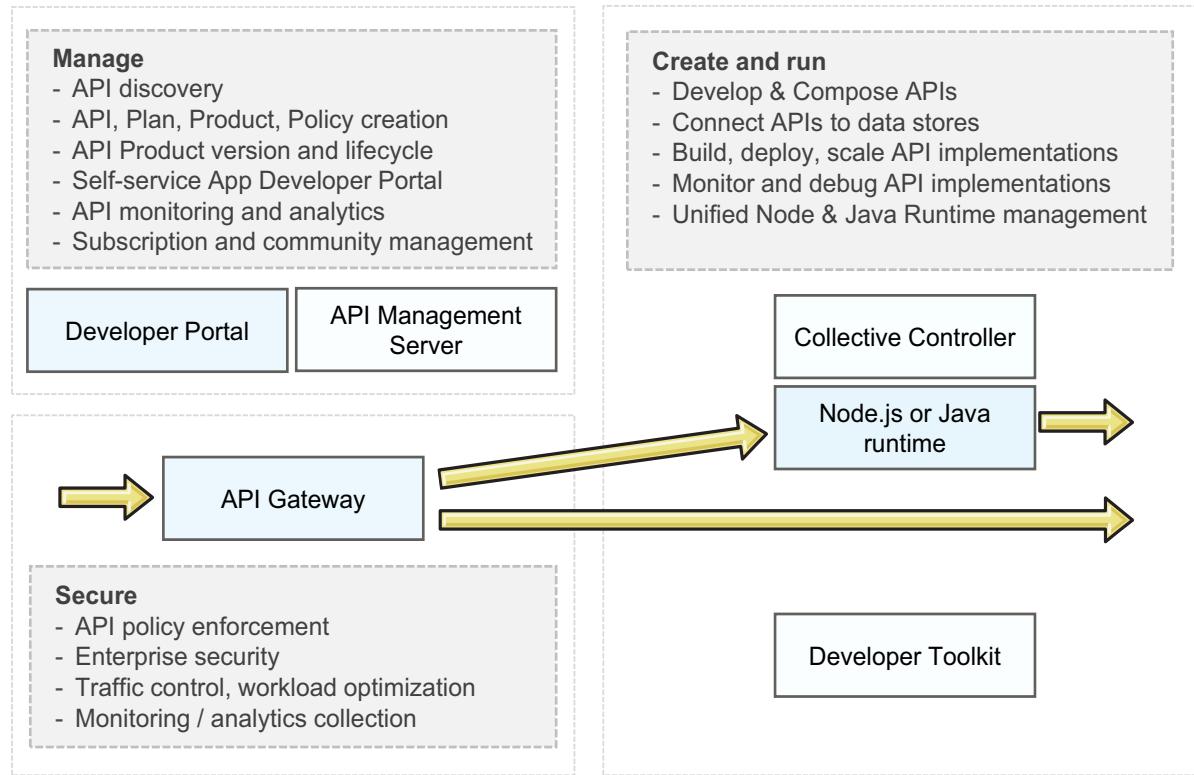
© Copyright IBM Corporation 2016

Figure 1-16. API Connect: Topology view

The topology view displays how the six components of IBM API Connect relate to one another.

1. The API developer (not shown) creates a Node.js LoopBack API application and defines an API interface in the Developer Toolkit. LoopBack is a model-driven API framework for Node.js, one of two supported runtime platforms for API implementations.
2. She deploys the API implementation to the Node.js compute runtime. This runtime is managed by an API Collective Controller, which in turn is controlled by the API Management Server. At the same time, she publishes the API with a Product and a Plan to the Management Server.
3. The Management server configures the API Gateway with the API definition and makes the API subscribeable on the Developer Portal.
4. A third-party app developer wants to use the API. He subscribes his client application with the API. After configuring his app, he invokes the API through the API Gateway.
5. Depending on the security and message processing policies, the API request is forwarded to the compute runtime, or it directly invokes an enterprise service, data source, or application.
6. The client app receives the result of the API call through the API Gateway.

## API Connect: Components by role



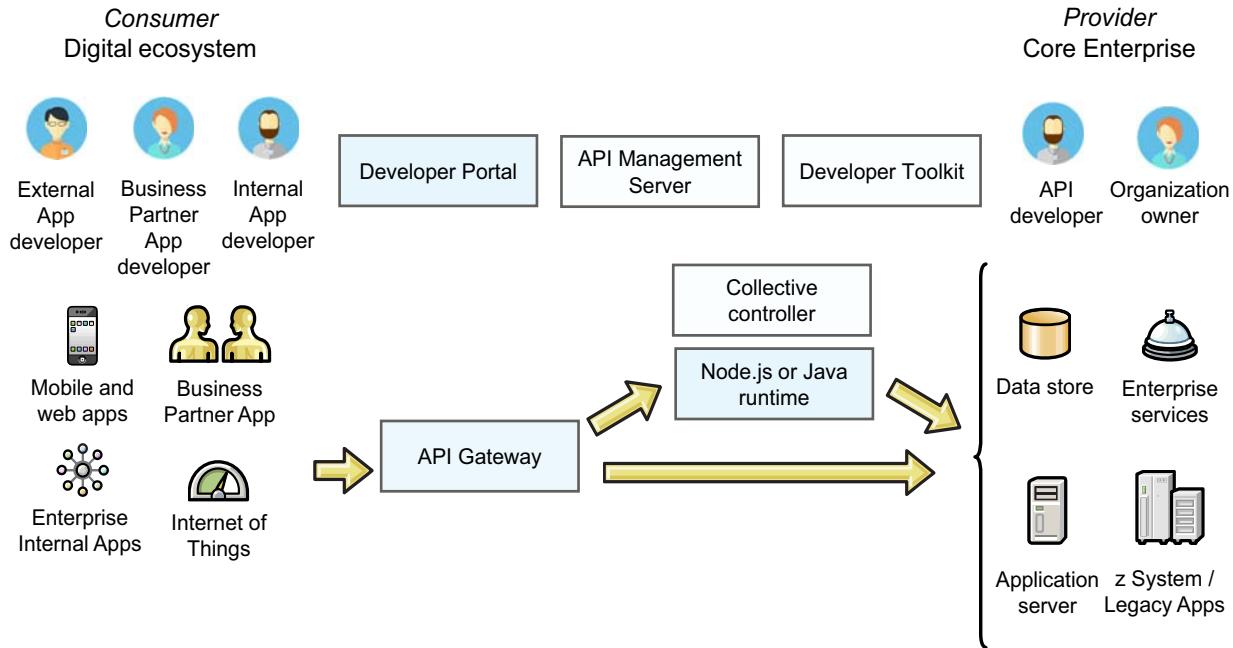
Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-17. API Connect: Components by role

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

## API Connect: Consumers and providers view



Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-18. API Connect: Consumers and providers view

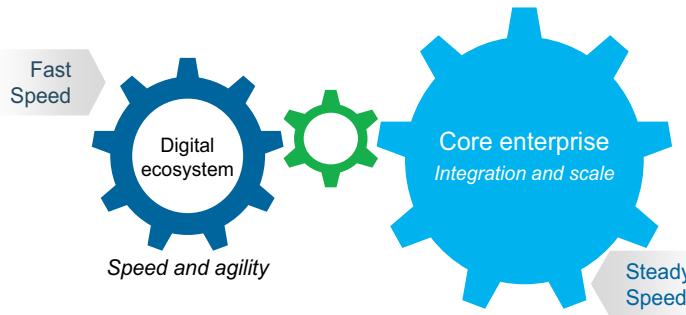
In this diagram, you examine the architectures in which API consumers and API providers interact with IBM API Connect.

The **digital ecosystem** is an architecture that focuses on supporting a wide range of clients. Mobile, web applications, business partner applications, enterprise applications, and the internet of things all represent different API consumers. Each client type has different security, access, and quality-of-service requirements. Each developer type has their own demands. For example, your internal and partner application developers might follow a standard invocation and data standard. However, your external application developers use a wide variety of standards and platforms.

The **core enterprise** represents the traditional, proven, and robust IT architecture. These systems focus on maintaining strong standards on data, messaging, and application platforms. They support a variety of data stores, enterprise services, application server platforms, and legacy applications.

The goal of API Connect is to expose or build a set of APIs to support a rich digital ecosystem. The existing rich set of applications in the core enterprise provide the implementation for the APIs.

## Digital ecosystem and enterprise systems



- The **digital ecosystem** focuses on building **systems of engagement**.
  - Decentralized systems that encourage peer-to-peer interactions.
- Focuses on **people and interactions**
- These systems make APIs available to consumers **outside** the company that self-subscribe to services.
- API consumers have **different** needs
- **Enterprise systems** focus on building **systems of record**.
  - Maintaining a single, consistent authority on information.
- Focuses on **data and processes**
- These systems make APIs available to organizations **inside** the same company.
- API consumers have **similar** needs

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-19. Digital ecosystem and enterprise systems

The **enterprise system** represents the traditional architecture that you find in medium and large companies. These companies build applications that focus on building **systems-of-record**: a philosophy of maintaining a single, consistent authority on information. For example, an enterprise application stores a set of customer records in a relational database with transactions. This database represents the authoritative "truth" on the customer record.

In a system-of-record, the focus is on building an architecture around business data and processes. The architecture assumes that the consumers of the business data and processes have similar needs. Therefore, enterprise systems focus on data-centric protocols such as SOAP web services. This design expects all of the API Consumers to adopt and support these complex data message formats.

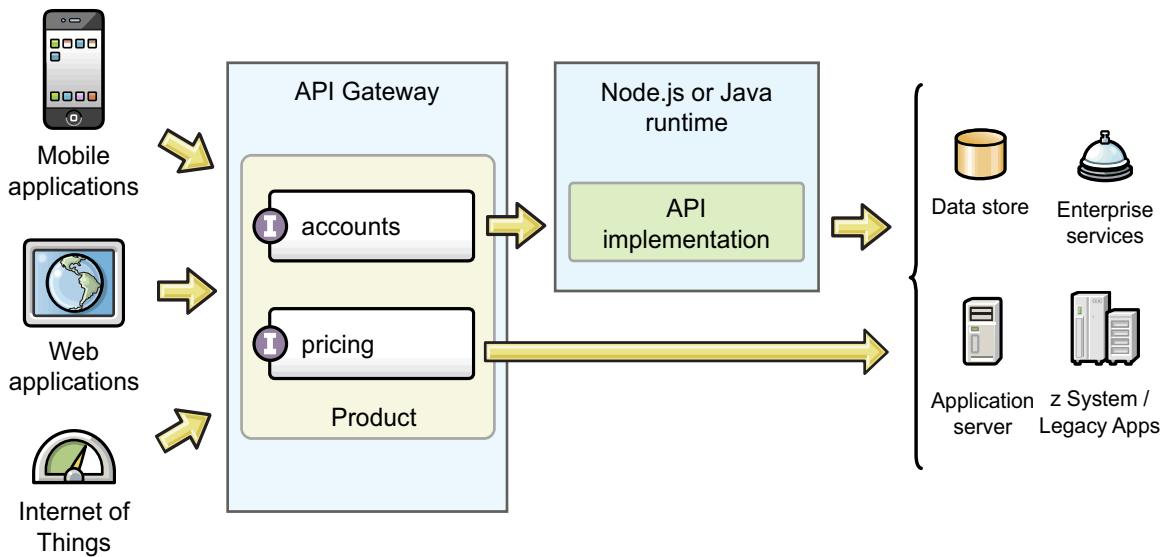
Although **enterprise systems** are robust and well-built, the architecture makes it difficult to build APIs for a range of consumers, each with different use cases and messaging formats. In contrast, a **digital ecosystem** that focuses on a **system-of-engagement** is better suited to support a range of consumer types.

In a **system-of-engagement** architecture, the focus is on building a decentralized system that encourages peer-to-peer interactions. These systems publish a collection of APIs that support a wide range of client types. The main focus is on making it easier to consume the API on a range of computer platforms.

In summary, both architecture types are important in a modern IT system. Your company must maintain an enterprise architecture to guarantee the integrity of your data, and also support a digital ecosystem to quickly support a range of client types from API consumers with different needs.

The gears in this diagram do not imply any coupling between the digital ecosystem and the core enterprise. It merely states that the two sides of your IT infrastructure changes at different speeds.

## API runtime message flow



Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-20. 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: Implement interaction 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 interaction 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 a proven, programming model that is familiar to enterprise application developers.
- The API Connect toolkit provides tools to build interaction APIs with the LoopBack framework on Node.js.
  - You can use your own development tools and processes to build interaction APIs on other languages and platforms.

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-21. Scenario 1: Implement interaction APIs

One of the main drivers for building interaction APIs is to support the demands from your customers. Your customers demand that your APIs support the latest computing standards, device types, and programming languages. Whereas enterprise systems maintain stable, uniform standards that change slowly, your digital ecosystem 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 interaction APIs with the enterprise languages that your developers understand, such as Java.

## Scenario 2: 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.

Introduction to IBM API Connect V5.0

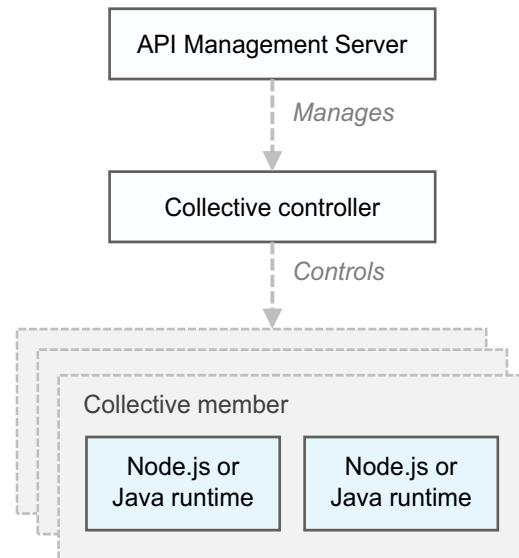
© Copyright IBM Corporation 2016

Figure 1-22. Scenario 2: Proxy existing APIs

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

## Manage API runtime environments with Collectives

- A collective is a group of multiple servers in a single management domain.
  - Within an API Connect environment, the collective deploys and runs API implementations and the micro gateway.
- Collectives provide:
  - Lightweight cluster support
  - A common point for automated deployment to multiple host systems
  - A secure, scalable, highly available operations point of control
  - A simple control point for monitoring operational status



[Introduction to IBM API Connect V5.0](#)

© Copyright IBM Corporation 2016

*Figure 1-23. Manage API runtime environments with Collectives*

A collective is a group of multiple servers from a single management domain. The API Connect collective controller manages member servers in the collective. Each collective member hosts a micro service, a server runtime environment for API implementations, or both. The server runtime environment for API implementations is known as a compute runtime.

## 1.3. Gateway options

## Gateway options

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

*Figure 1-24. Gateway options*

## Topics

- API Connect Overview
- Topology and architecture
-  Gateway options
- Software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-25. 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 provide 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** providing basic functions to empower developers and supports single department projects starting on their API journey.
  - Option to upgrade to a DataPower Gateway to meet advanced, enterprise-grade API Gateway needs.

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-26. 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 (Enterprise API 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/cluster
- Micro gateway (Basic API gateway)
  - Built for developers and single API projects
  - Programmable with JavaScript, built on Node.js
  - Embedded into native developer experience
  - Basic set of security and traffic management features
  - Supports a single catalog per instance/cluster

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-27. Gateway features

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

## 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 native developer experience	No	Yes
Purpose-built, DMZ-ready, secure gateway platform	Yes (2)	No
Multi-channel gateway: mobile, web, API, B2B, SOA	Yes	No
Enterprise-grade performance	Yes (3)	No
Reverse proxy, SSL/TLS termination	Yes	No (4)
Built-in policies	Advanced (5)	Standard
Rate limit	Yes	Yes
Any-to-any message transformation	Advanced	Yes (6)
Database connectivity, IBM System z connectivity	Yes	Yes (7)
Identity and Access Management system integration	Advanced	Yes (8)

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-28. Gateway capabilities (1 of 2)

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

## Gateway capabilities (2 of 2)

Capability	DataPower Gateway	Micro Gateway
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 (9)	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 (10)	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 (11)	No

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-29. Gateway capabilities (2 of 2)

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

## 1.4. Software and hardware prerequisites

## Software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

*Figure 1-30. Software and hardware prerequisites*

## Topics

- API Connect Overview
  - Topology and architecture
  - Gateway options
-  Software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-31. Topics

## API Connect: Requirements

- Review the latest 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)

Continuous Delivery Product - Long Term Support Release ⓘ

**IBM API Connect 5.0 Enterprise**

Detailed System Requirements

Report filters

Available Reports

5.0 Enterprise

Operating Systems	Hypervisors	Prerequisites	Supported Software												
Linux	Mac OS	Windows													
<b>Linux</b> <table border="1"> <thead> <tr> <th>Operating System</th> <th>Operating System Minimum</th> <th>Hardware</th> <th>Components</th> <th>Notes</th> <th>Details</th> </tr> </thead> <tbody> <tr> <td>Red Hat Enterprise Linux (RHEL) Server 7</td> <td>Base</td> <td>IBM</td> <td>           64-Exploit, 64-Tolerate            64-Exploit, 64-Tolerate            32, 64-Exploit, 64-Tolerate         </td> <td>No</td> <td></td> </tr> </tbody> </table>				Operating System	Operating System Minimum	Hardware	Components	Notes	Details	Red Hat Enterprise Linux (RHEL) Server 7	Base	IBM	64-Exploit, 64-Tolerate 64-Exploit, 64-Tolerate 32, 64-Exploit, 64-Tolerate	No	
Operating System	Operating System Minimum	Hardware	Components	Notes	Details										
Red Hat Enterprise Linux (RHEL) Server 7	Base	IBM	64-Exploit, 64-Tolerate 64-Exploit, 64-Tolerate 32, 64-Exploit, 64-Tolerate	No											

Filter

Server Component Support

- ✓ API Connect Collective Controller
- ✓ API Connect Collective Member
- ✓ Micro Gateway

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-32. API Connect: Requirements

## Unit summary

- Explain the digital ecosystem and enterprise core architectures
- Identify the components in the IBM API Connect V5.0 solution
- Identify the API creation, run, manage, and secure features
- Identify the software and hardware prerequisites

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

*Figure 1-33. 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 a compute runtime?
  - A. It is an application runtime environment for API implementations.
  - B. It is a component that is managed by a Collective Controller.
  - C. There are two types of compute runtimes: Node.js, and Java.
  - 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. TLS/SSL termination support.



Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-34. 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 a compute runtime?
  - A. It is an application runtime environment for API implementations.
  - B. It is a component that is managed by a Collective Controller.
  - C. There are two types of compute runtimes: Node.js, and Java.
  - 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. TLS/SSL termination support.  
The answer is D.

Introduction to IBM API Connect V5.0

© Copyright IBM Corporation 2016

Figure 1-35. Review answers

---

# Unit 2. Create an API with the API Connect toolkit

## Estimated time

00:45

## Overview

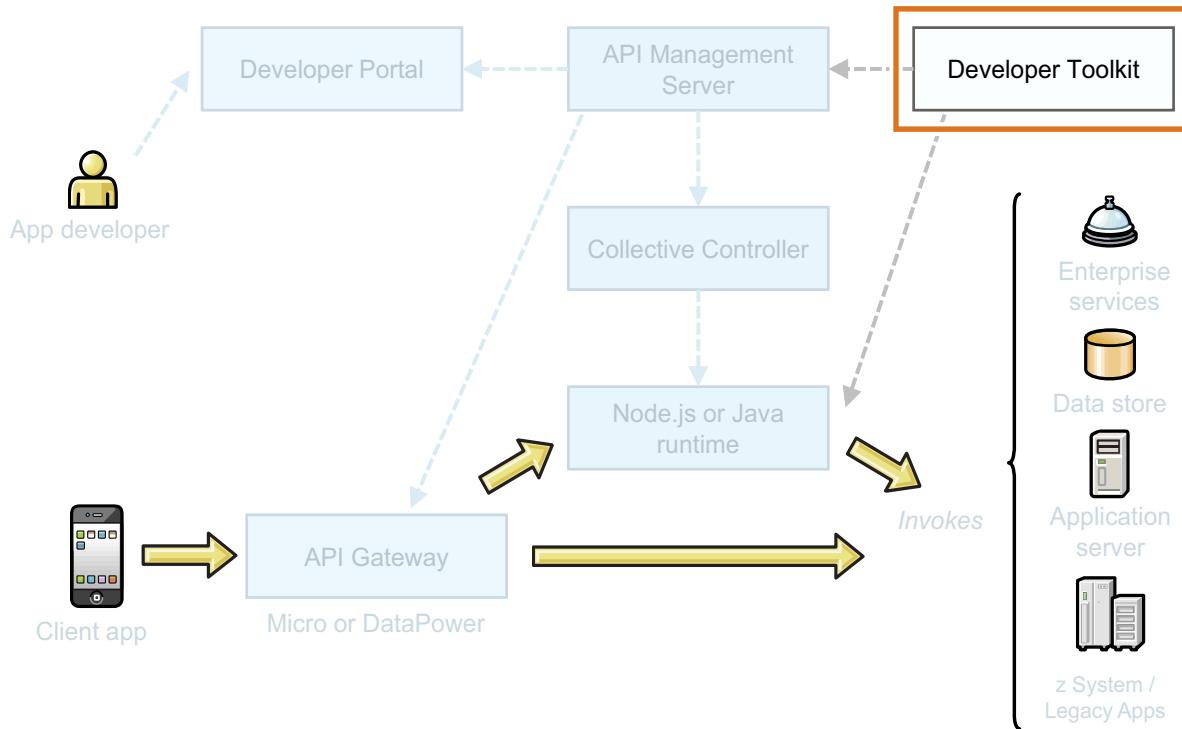
In this unit, you learn how to create APIs and LoopBack applications with the API Connect toolkit. LoopBack is a Node.js application framework for REST APIs. You learn how to install the API Connect toolkit on your own workstation. You design and develop APIs with the apic command-line utility and the API Designer web application from the toolkit.

## Unit objectives

- Explain the concept of interaction APIs
- Identify the features of the API Connect toolkit
- Identify the structure of a LoopBack Node.js application
- Explain how to implement an API with the LoopBack framework
- Identify the features of the API Editor and API Explorer



## API Connect: Topology view



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-2. API Connect: Topology view

The topology view displays how the six components of IBM API Connect relate to one another.

This unit focuses on the LoopBack generation features in the `apic` command-line utility, and the API Designer web application. These applications are part of the API Connect Developer Toolkit.

## 2.1. Introduction to interaction APIs and LoopBack

## Introduction to interaction APIs and LoopBack

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

*Figure 2-3. Introduction to interaction APIs and LoopBack*

## Topics

### Introduction to interaction APIs and LoopBack

- The IBM API Connect toolkit
- Implement an interaction API with a LoopBack application

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-4. Topics

## API terminology

- What is an **API**?
  - An **application programming interface** is a collection of remote service operations that you make available to API consumers.
  - In **IBM API Connect**, the API is a collection of REST service operations.
  
- 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 2-5. 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**.

In between the API consumer and the API provider is the **API Gateway**. 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 guarantees, 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 of 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 actual cost of the product is saved in an existing database.
- To calculate the price on demand, define an **interaction API** that **invokes** an **existing API** in its implementation.

Create an API with the API Connect toolkit

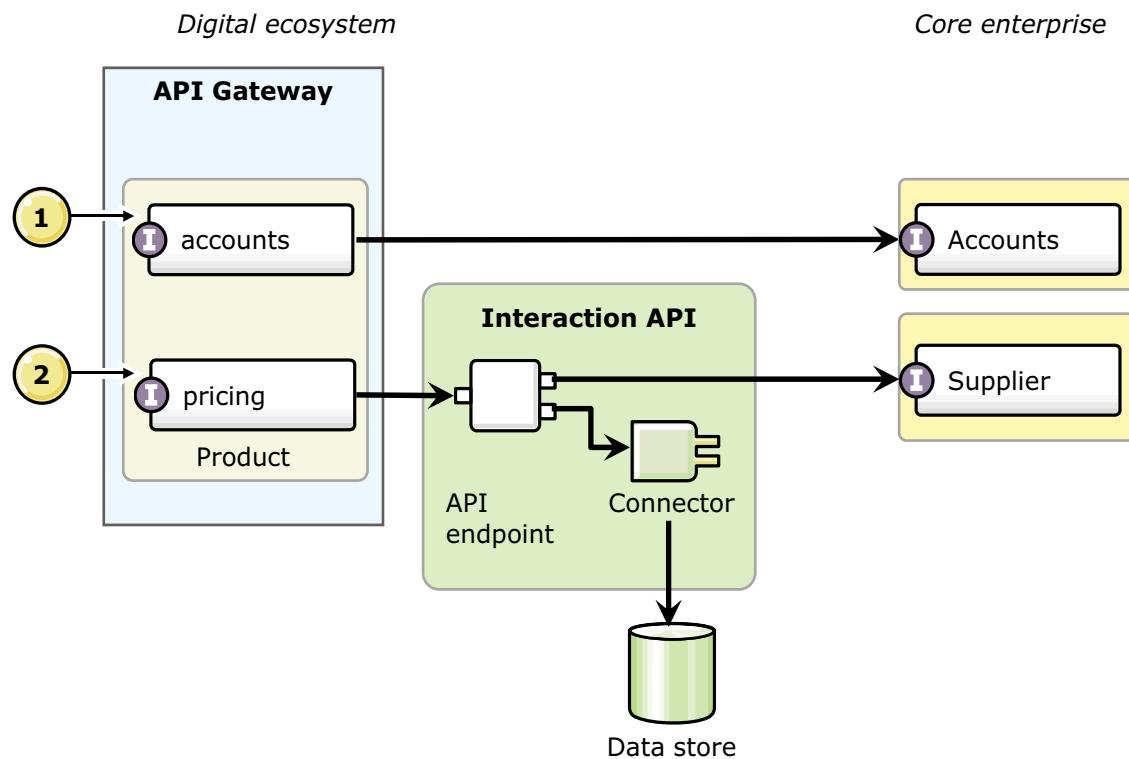
© Copyright IBM Corporation 2016

Figure 2-6. 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 partners 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 calculates the prices of a product based on inventory levels. However, the pricing API requires the caller to provide an internal product and warehouse identifier. To abstract these details from your API consumers, you build an interaction API that invokes the existing API in its implementation.

## Message flows in IBM API Connect



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

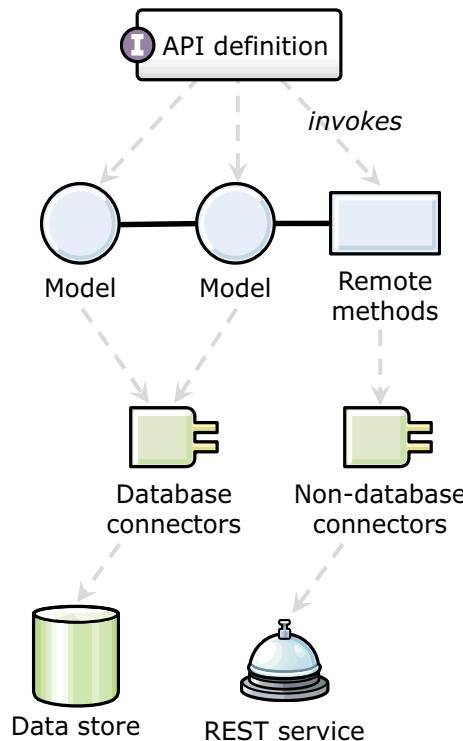
Figure 2-7. Message flows in IBM API Connect

This diagram illustrates how interaction 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 **interaction API**. The interaction API invokes one or more existing APIs and data sources. It manipulates the data from these sources with its own logic. The purpose of an interaction API is to reuse existing sources of data and services with new business logic.

## What is LoopBack?

- LoopBack is a Node.js framework for interaction 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 to a data access method to a model.



[Create an API with the API Connect toolkit](#)

© Copyright IBM Corporation 2016

Figure 2-8. 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 interaction APIs: standalone APIs that are used entirely within the interaction services layer. In the following slide, you examine how interaction APIs integrate with existing services at the system API layer.

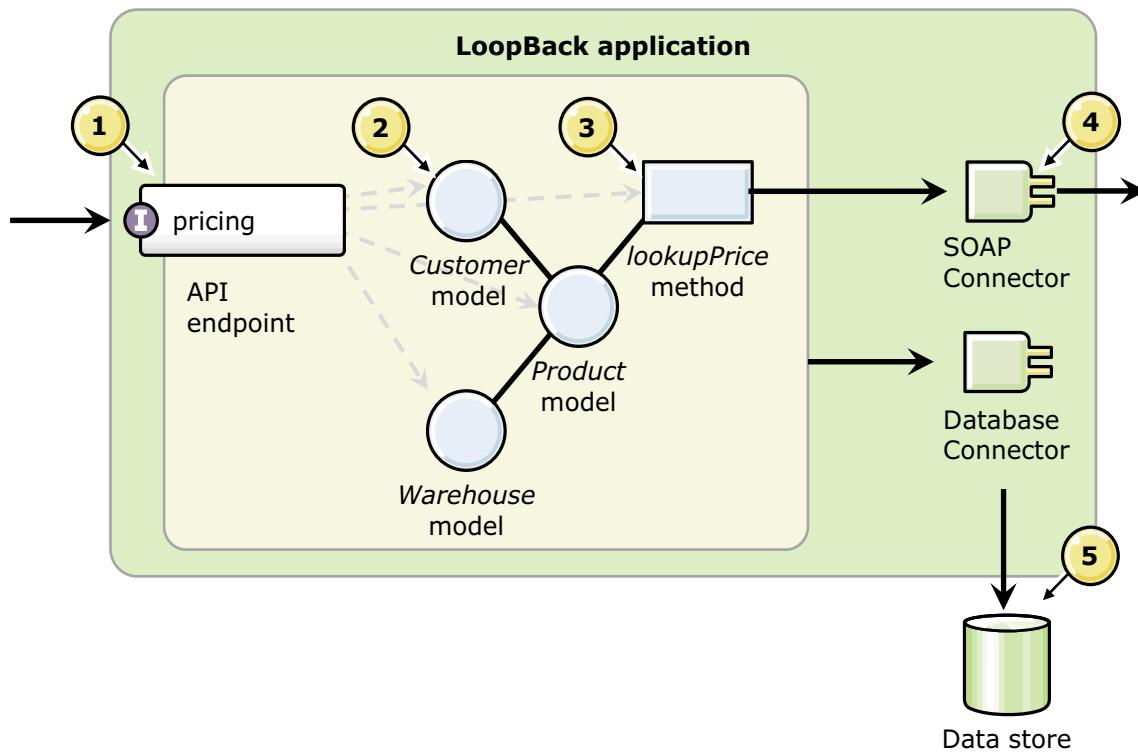
With LoopBack, you define business models, properties, and relationships through command-line tools. The LoopBack framework creates a set of pre-defined operations to create, retrieve, update, and delete models. The framework also persists the model data to a data source through a database connector.

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.

## Interaction API with a Node.js LoopBack application



Create an API with the API Connect toolkit

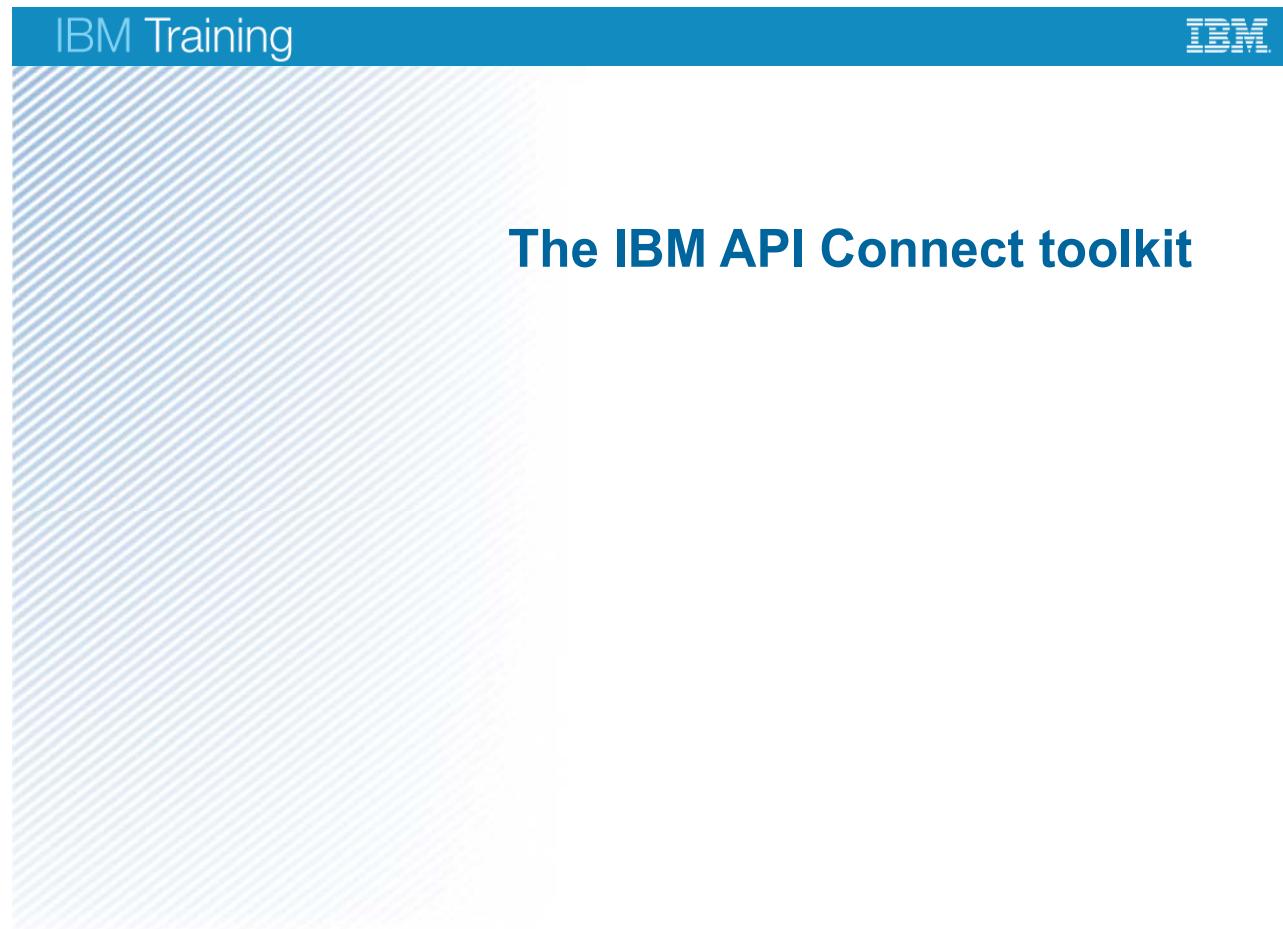
© Copyright IBM Corporation 2016

Figure 2-9. Interaction API with a Node.js LoopBack application

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

1. The API definition Swagger 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 interaction APIs. LoopBack provides a set of **non-database connectors** to make it easier for you to invoke 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.

## 2.2. The IBM API Connect toolkit



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

*Figure 2-10. The IBM API Connect toolkit*

## Topics

- Introduction to interaction APIs and LoopBack
- ▶ The IBM API Connect toolkit
- Implement an interaction API with a LoopBack application

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-11. Topics

## What is the IBM API Connect toolkit?

- You build **APIs** 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.

[Create an API with the API Connect toolkit](#)

© Copyright IBM Corporation 2016

*Figure 2-12. 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.

The API Connect toolkit is available for developers on all editions of API Connect: essential, professional, and enterprise. You can install the API Connect toolkit locally on your own workstation even if you run the cloud-enabled Bluemix version of API Connect.

## How do you install the toolkit?

- **Review the latest installation instructions:**
  - <https://developer.ibm.com/apiconnect/getting-started/>
- **Install the prerequisite software** on your workstation:
  - Node.js
  - C++ compiler (version depends on your operating system)
  - Python
  - npm
- To install the **IBM API Connect toolkit**, use the Node package manager:

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

Figure 2-13. How do you install the toolkit?

Before you continue, open the latest installation instructions from the IBM Developer site. Confirm that you have the correct version of Node.js, C++ compiler, Python, and npm 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. This scenario is useful if your workstation does not have access to the NPM repository. Review the API Connect documentation for instructions on how to install the toolkit from an API Manager.

## Operating system requirements

- 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)

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-14. Operating system requirements

See <https://developer.ibm.com/apiconnect/getting-started/> for the specific versions of the software listed above.

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

## 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 Swagger documents.
  - 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
```

Create an API with the API Connect toolkit

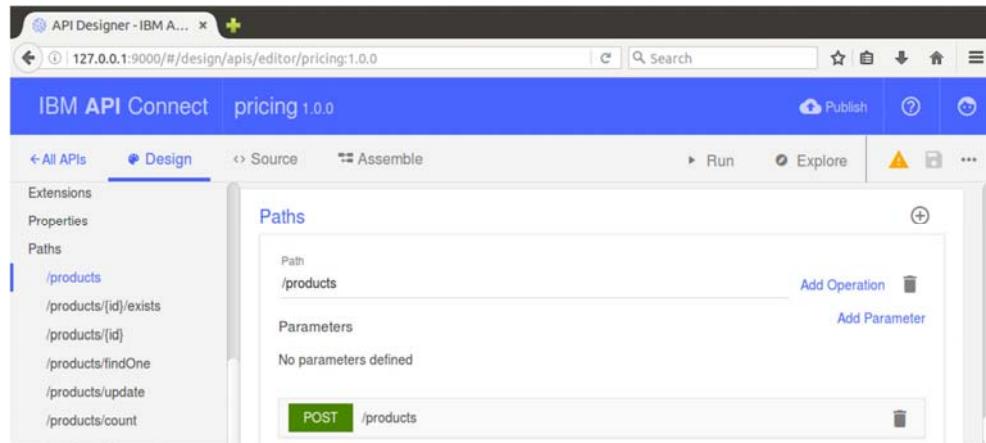
© Copyright IBM Corporation 2016

Figure 2-15. API Connect Toolkit: The apic command



## API Connect Toolkit: The API designer web application

- The **API Designer** is an API **design**, **configuration**, and **testing** application.
  - No internet connection required: the API designer runs from your local workstation.
  - You interact with the API Designer through your web browser.
- You can also deploy Node.js LoopBack applications to a Bluemix compute runtime in API Designer.



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-16. API Connect Toolkit: The API designer web application

The API Designer is a web browser application that your computer hosts. You launch 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 **interaction 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.

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-17. How to create APIs in API Connect

For a detailed discussion and demonstration on how to invoke an existing API in API Connect, see the "Define an API in API Designer" unit.

In order 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.

## 2.3. Implement an interaction API with a LoopBack application

## Implement an interaction API with a LoopBack application

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

*Figure 2-18. Implement an interaction API with a LoopBack application*

## Topics

- Introduction to interaction APIs and LoopBack
  - The IBM API Connect toolkit
-  Implement an interaction API with a LoopBack application

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-19. Topics

## Instructor demonstration

- In this scenario, you implement an interaction 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 interaction 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. Start the LoopBack application and micro gateway.
9. Test the interaction API with the API Explorer.

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-20. Instructor demonstration

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

## 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
? What kind of application do you have in mind?
    empty-server API, without any configured models or data
sources
```

3. Change directory into your LoopBack application.

```
$ cd pricing
```

*Figure 2-21. Step 1: Generate a 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 perform the rest of the steps in the LoopBack directory. The *apic* commands do not work if you call them in another directory.

## Step 2: Define a data source

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

### 1. Create a LoopBack data source.

```
$ 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 2-22. Step 2: Define a data source

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

## Step 3: 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 2-23. Step 3: Create models, properties, and relationships

## Step 4: Implement Interaction API logic

- Implement the message processing logic in the interaction API in Node.js.

### 1. Implement the message processing logic in the LoopBack model.

```
module.exports = function(Product) {
  var priceService;

  Product.on('attached', function() {
    priceService = Product.app.dataSources.priceREST;
  });

  Product.lookupPrice = function() {
    priceService.price.apply(priceService, arguments);
  }

  Product.remoteMethod('lookupPrice', {
    description: 'Calculate price for specified product',
    accepts: [{ arg: 'productId', type: 'number' }],
    returns: { arg: 'price', root: true },
    http: { verb: 'get' }
  });
}
```

[Create an API with the API Connect toolkit](#)

© Copyright IBM Corporation 2016

Figure 2-24. Step 4: Implement Interaction API logic

In this example, you defined a remote method 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.

## 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/localuser/projects/pricing/definitions/pricing.yaml
swagger description
```

*Figure 2-25. 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 is 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. Swagger V2.0 is an open-source API definition format. One of the representations of a Swagger document is the text-based "Yet Another Markup Language" format.

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 invoke **apic loopback:refresh** in order 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.

1. Start the API Designer application.

```
$ apic edit
```

2. Sign in to the Designer with your Bluemix account.



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

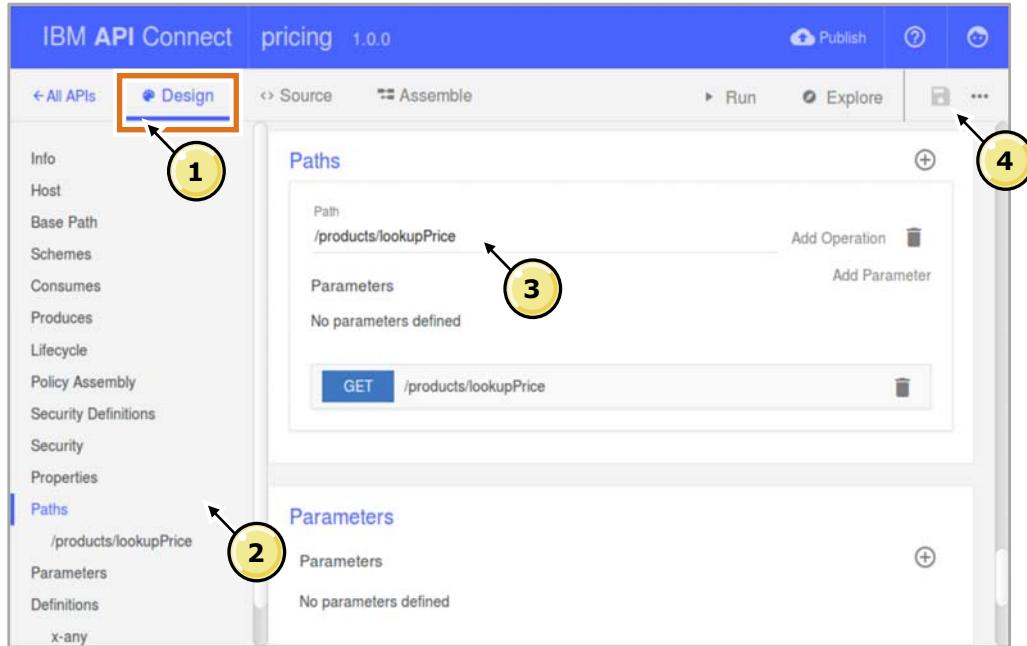
Figure 2-26. Step 6: Start the API Designer application

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



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-27. Step 7: Review the API definition file with API Designer

1. 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 correspond to the sections of a Swagger V2.0 document. Swagger is an industry-standard specification for defining the structure of REST (Representational State Transfer) APIs.
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.
4. After you made any edits to the API definition file, select the **Save** command to write your changes to the Swagger document.

## Step 8: Start the LoopBack application and micro gateway

- To test your interaction API, you must start two components:
  - The **LoopBack application** implements the interaction API.
  - The **micro gateway** makes the interaction 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:



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-28. Step 8: Start the LoopBack application and micro gateway

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.

## 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 limiting, access control, message mapping, and other policies before it invokes the LoopBack application.

*Figure 2-29. Relationship between API gateway and LoopBack*

Why do you need to start two components?

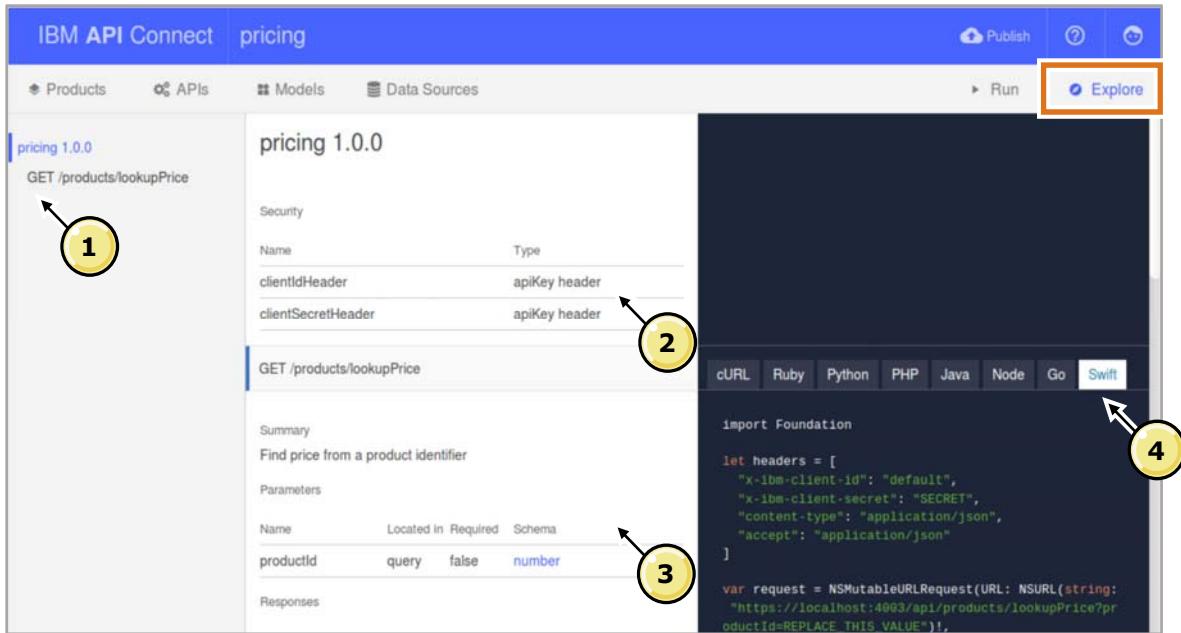
The LoopBack application is the implementation for your API.

When you invoke 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 named **product** with two properties: **name** and **price**. When you call GET /api/products, you retrieve a JSON object with a name and price. No coding is required to build these APIs.

In short, these 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.

## Step 9: Test the interaction API with the API Explorer

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



Create an API with the API Connect toolkit

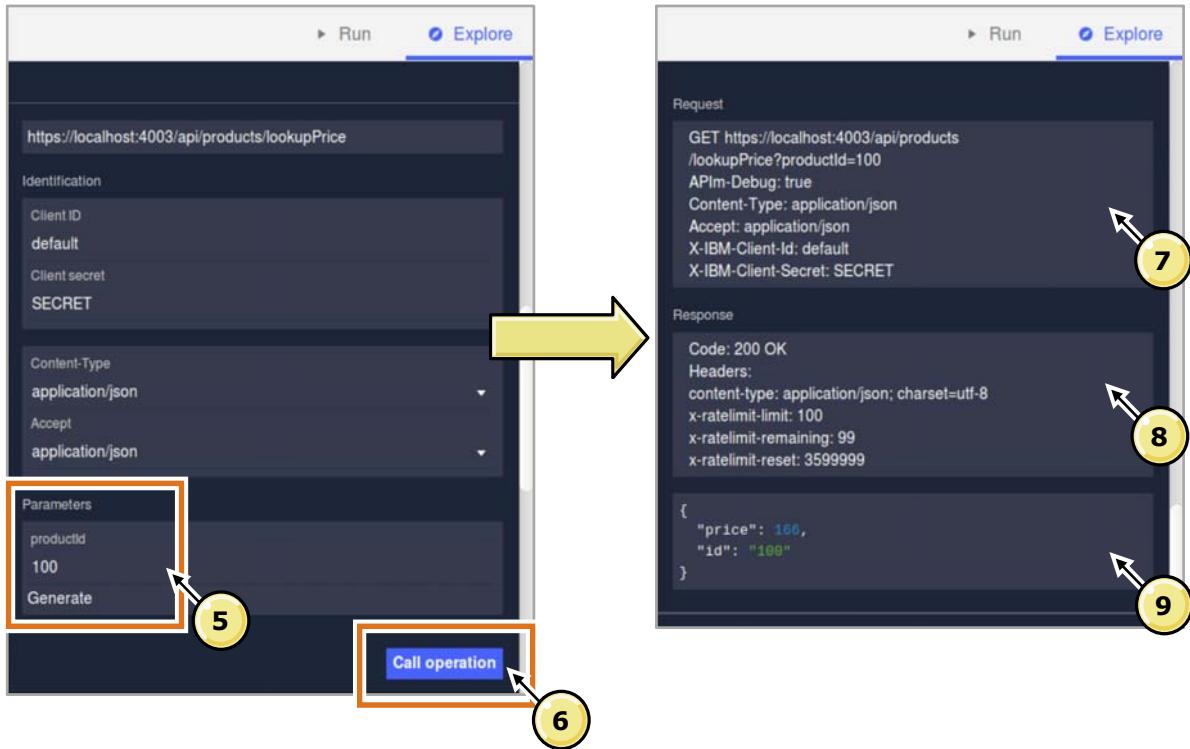
© Copyright IBM Corporation 2016

Figure 2-30. Step 9: Test the interaction 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 name, version, and operations.
2. At the top of the API details view, the API Explorer lists properties that apply to all API operations. In this example, the API definition expects `clientIdHeader` and `clientSecretHeader` fields in the HTTP request message header.
3. For each API operation, the details column displays the input and output parameters. In this example, the `GET /products/lookupPrice` API operation has an optional parameter that represents the product identifier number.
4. 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.

## Example: Test API operations in Explorer



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-31. Example: Test API operations in Explorer

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

5. 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 100. Optionally, you can select **generate** to create random values based on the parameter data type.
6. Select **Call operation** to make an HTTP or HTTPS request to the API operation.
7. Scroll down further to review the request and response messages from the API call.
8. The response message header displays the HTTP status code and response header values.
9. The last section displays the HTTP response message body. In this example, the lookupPrice API operation returns an JSON object with a price and a product identifier value.

## Unit summary

- Explain the concept of interaction APIs
- Identify the features of the API Connect toolkit
- Identify the structure of a LoopBack Node.js application
- Explain how to implement an API with the LoopBack framework
- Identify the features of the API Editor and API Explorer

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

*Figure 2-32. Unit summary*

## Review questions

1. **True or False:** You must have an IBM Bluemix account to sign on to the API Designer web application.
2. **True or False:** You deploy your LoopBack application to the API Gateway.
3. **True or False:** You must create a LoopBack application to edit the API interface definition in the API Designer web application.
4. **True or False:** In IBM API Connect, a LoopBack application is a way to implement an interaction API.
5. **True or False:** To install the API Connect toolkit, install a Node.js runtime and install the apiconnect npm package.



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-33. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True. To open the API Designer web application, you must log in with your IBM Bluemix account.
2. False. You deploy your LoopBack application to your own server, or as an IBM Bluemix service. You cannot deploy LoopBack applications to the API Gateway.
3. False. You can create a Swagger API definition in API Designer without creating a LoopBack application.
4. True. LoopBack applications is an option for implementing interaction APIs.
5. False. You must install the correct Python, C++ compiler toolchain, npm, and operating system software before you install the apiconnect package.



## Exercise: Introduction to IBM API Connect

### Lab 1

Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

Figure 2-35. Exercise: Introduction to IBM API Connect

## Exercise objectives

- Create a simple LoopBack application
- Create a REST API definition
- Test a REST API
- Publish an API to the Developer Portal



Create an API with the API Connect toolkit

© Copyright IBM Corporation 2016

*Figure 2-36. Exercise objectives*

---

# Unit 3. Define an API with the API Designer

## Estimated time

01:00

## Overview

In this unit, you learn how to define and edit the API interface with the API Designer. Learn how the Swagger 2.0 specification is used for defining and displaying API interfaces. Explore how the API Designer is used to create a REST API definition that calls an existing system API.

## How you will check your progress

- Lab exercise
- Review questions

## Unit objectives

- Explain the concept of an API definition
- Explain the role of the Swagger specification in defining REST API interfaces
- Describe the structure of an API definition
- Design a REST API interface with the API Designer
- Explain the purpose of the assembly editor in the API Designer
- Create a REST API definition that invokes an existing system API

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-1. Unit objectives

## User story and role when defining APIs

- User story: As an API Developer, you want to create an API definition with all its operations
  - Role: API Developer



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-2. User story and role when defining APIs

## 3.1. Define the API with Swagger 2.0

## Define the API with Swagger 2.0

Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-3. Define the API with Swagger 2.0*

## Topics

### Define the API with Swagger 2.0

- Features of the API Designer

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-4. Topics

## What is an API definition?

An API definition is composed of paths, and can be one of these types:

### 1. REST API definition

- A defined set of interactions that uses the HTTP protocol
- Typically uses the JSON or XML data format for messages
- Paths are the routes through which users access REST APIs
- Configure your API definition by using the API Designer or by writing a Swagger definition and publishing it

### 2. SOAP API definition

- Based on an existing Web Services Description Language (WSDL) file
- Create SOAP API definitions with the command line or on API Manager

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

*Figure 3-5. What is an API definition?*

An API is a contract between a producer and a consumer.

An API definition is composed of paths, and can be one of these types:

### 1. REST API definition

### 2. SOAP API definition

This course focuses on configuring REST API definitions by using the API Designer that is part of the API Connect toolkit.

## What is the Swagger OpenAPI specification?

- Swagger OpenAPI is an open source specification that defines APIs in a language-independent manner
- The Swagger OpenAPI specification defines a standard way to describe and document RESTful APIs
  - Licensed under the Apache license
  - Open source software
  - Extensible
- The Swagger API interface definition is a standard way for defining REST APIs
  - YAML source code, a superset of JSON, can be used to represent a Swagger specification file
- API Connect follows the Swagger OpenAPI specification
- API Connect adds its own extensions to the specification to define:
  - The message processing flow
  - The API product
  - The deployment environment

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

*Figure 3-6. What is the Swagger OpenAPI specification?*

When properly defined in the Swagger OpenAPI format, a user can understand and interact with the remote service without access to source code or through network traffic inspection. Similar to what interfaces have done for lower-level programming, Swagger removes the guesswork in calling the service.

The Swagger OpenAPI format makes it easy for the developer to see the REST operations, parameters, request, and response messages. The Swagger OpenAPI format also shows the possible HTTP return codes for successful and the HTTP error codes for unsuccessful REST calls. Swagger OpenAPI also defines the data schemas for the data that is passed by the REST calls.

Swagger OpenAPI defines the REST operations to read, insert, update, and delete from a remote service.

YAML is a text-based, easily readable file structure for describing APIs according to the Swagger OpenAPI specification.

## REST operations

### REST HTTP methods

- GET
  - Request the specific resource
  - Response message returns data in the JSON format, in many cases
- POST
  - Used to create a new resource
- PUT
  - Used to set values
- DELETE
  - Delete a specific resource
- HEAD
  - Used to determine whether a resource is available
- OPTIONS
  - Used to retrieve the available HTTP REST operations for a resource

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

*Figure 3-7. REST operations*

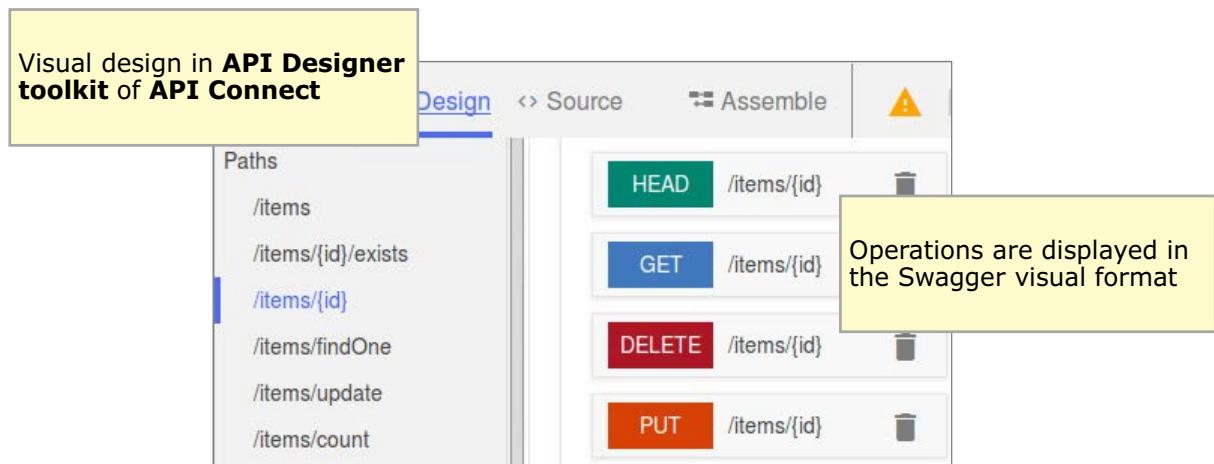
REST APIs use HTTP as the transport mechanism.

The REST API provides a number of common request operations for data read, add, update, and delete functions.

Strictly speaking the REST operations include GET, POST, PUT, and DELETE, since REST can be used with other transport protocols other than HTTP.

## How API Connect uses the Swagger specification

- Provides language-independent syntax for API interface definition and visualization
- Captures API implementation details
  - Service endpoints
  - Message flow
  - A packaging description for grouping APIs and deployment environments



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-8. How API Connect uses the Swagger specification

The example shows the standard user interface for REST APIs in the Swagger V2.0 display format in the API Designer editor.

In the example, you see some of the standard REST operations displayed in the Swagger format.

## Specification format

- The Swagger specification for describing the RESTful API can be represented in JSON or YAML files
- The API Designer toolkit of API Connect uses the YAML format for defining APIs
  - YAML is easier for developers to read and edit than the JSON format
  - Case-sensitive field names
  - Uses a single file to define the API. For example, `hello-world.yaml`

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-9. Specification format

The API Designer toolkit of API Connect uses the YAML format to define APIs in accordance with the Swagger specification.

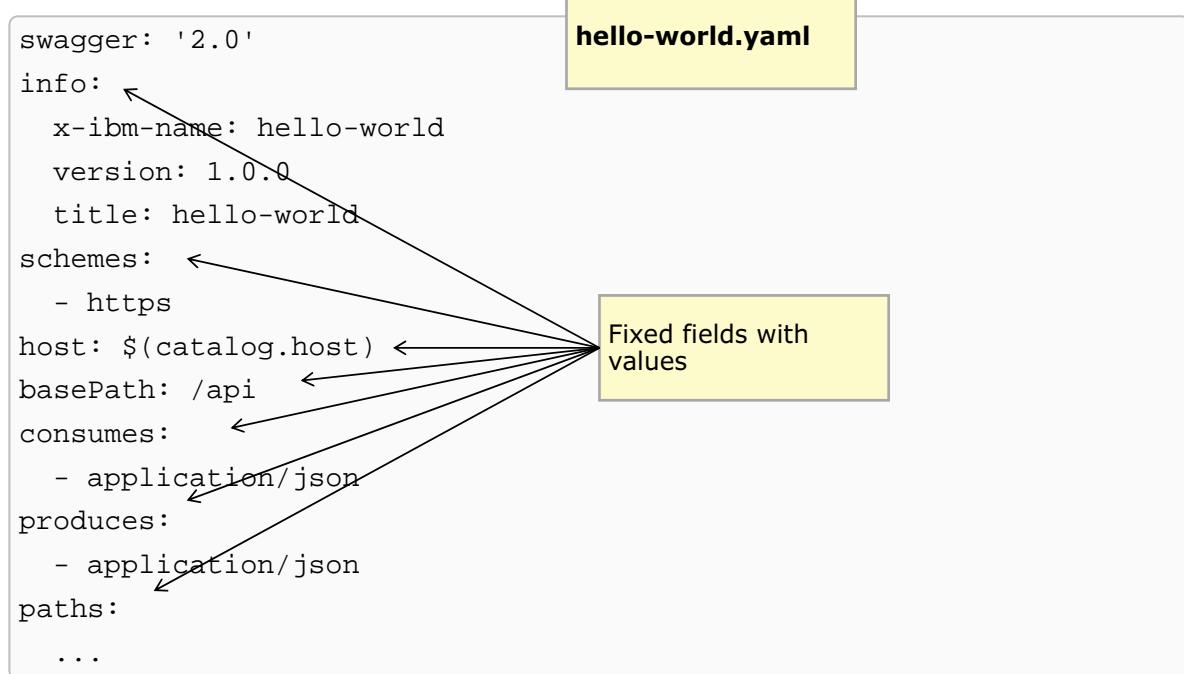
The Swagger representation of the API is made up of a single file.

However, the specification allows for parts of the definition to be split into a separate files.

These files can be referenced by using the `$ref` property that allows for an external definition of the path object.

## File structure

- The Swagger object is the root document object for the API specification



Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-10. File structure*

The Swagger object is the root document object for the API specification.

The API specification includes a number of fixed fields such as swagger, info, schemes, host, basePath, consumes, produces, and paths.

Each of these fixed fields is a name-value pair.

The required fields are swagger, info, and paths.

The swagger field specifies the Swagger specification that is being used. The value must be 2.0

The info field specifies metadata about the API.

The paths object specifies the available paths and operations for the API.

For more information, see <http://swagger.io/specification/>

## 3.2. Features of the API Designer

## Features of the API Designer

Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-11. Features of the API Designer*

## Topics

- Define the API with Swagger 2.0
- ▶ Features of the API Designer

Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-12. Topics*

## What is the API Designer?

- **API Designer** is a graphical API definition editor that runs in a web browser
- The Design view organizes the API definition into categories

Categories map to the fields defined by the Swagger OpenAPI specification

	Info	Info
Title	hello-world	
Name	hello-world	
Version	1.0.0	

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

Figure 3-13. What is the API Designer?

The API Designer toolkit is part of the API Connect product that can be used to design your API workflow.

The API Designer can be used for designing, editing, and testing your APIs on your local workstation. The API Designer toolkit can be installed from the website  
<https://www.npmjs.com/package/apiconnect>

With the API Designer toolkit, you can work offline to:

- Generate API interface definitions
- Edit the API in the Design view
- Write API specs in YAML (yet another markup language) in the Source view.
- Test the API to ensure that it is defined and implemented correctly.



## API Designer Source view

- The Source view displays the API definition in the Swagger YAML format

```

1  swagger: '2.0'
2  info:
3    x-ibm-name: hello-world
4    version: 1.0.0
5    title: hello-world
6    schemes:
7      - https
8    host: ${catalog.host}
9    basePath: /api
10   consumes:
11     - application/json
12   produces:
13     - application/json
14   securityDefinitions:
15     clientIdHeader:
16       type: apiKey
17       in: header
18       name: X-IBM-Client-Id
19     clientSecretHeader:
20       in: header
21       name: X-IBM-Client-Secret
22       type: apiKey
23   security:
24     - clientIdHeader: []
25     clientSecretHeader: []
26   x-ibm-configuration:
27     testable: true

```

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

Figure 3-14. API Designer Source view

The Source view displays the API definition as a Swagger-specification YAML file.

YAML is a text-based, easily readable file structure for describing APIs according to the Swagger specification. The Swagger specification also lets you specify APIs in JSON format, but developers often prefer working with the more readable YAML format source code.

The Source view of the API Designer includes an editor where YAML source code can be viewed or typed.

The editor automatically parses and syntax checks all changes that are made to the YAML specification and IBM extensions.

The editor validates the source code and can display warnings or errors when incorrect syntax is typed.

The API Designer synchronizes the code between the Source view and the Design view of the editor.



## Specifying an API operation in the API Designer visual view

GET	/notes/{id}			
Summary		Operation ID		
Find a model instance by id from the data source.		note.findById		
Parameters <a href="#">Add Parameter</a>				
Name	Located In	Description	Required	Type
id	Path ▾	Model id	<input checked="" type="checkbox"/>	string ▾
filter	Query ▾	Filter defining fields and	<input type="checkbox"/>	string ▾
Responses <a href="#">Add Response</a>				
Status Code	Description		Schema	
200	Request was successful		note ▾	

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

Figure 3-15. Specifying an API operation in the API Designer visual view

The API Designer Design view displays the REST operations in a visual format.

The example shows a **GET** operation that is followed by a path and a parameter.

Any parameters that are used in the operation are shown, along with the data type of the parameter, and whether it is required.

Expected HTTP response codes for successful calls are also shown.

The API Designer allows you to toggle between displaying the visual mode or the source mode in the browser window.



## Reference schema definitions from operations

- Reference a type that is defined as a schema definition

The screenshot shows the IBM API Designer interface for defining an API operation. The operation is a **POST** to the endpoint `/notes`. A tag named **note** is selected. The **Summary** field contains the text: "Create a new instance of the model and persist it". The **Operation ID** is set to `note.create`. In the **Parameters** section, there is one parameter named **data**, which is located in the **Body** and has a description of "Model instance data". The **Type** dropdown for this parameter is set to **note**, which is highlighted with a red box. There is also a small trash can icon next to the type dropdown.

[Define an API with the API Designer](#)

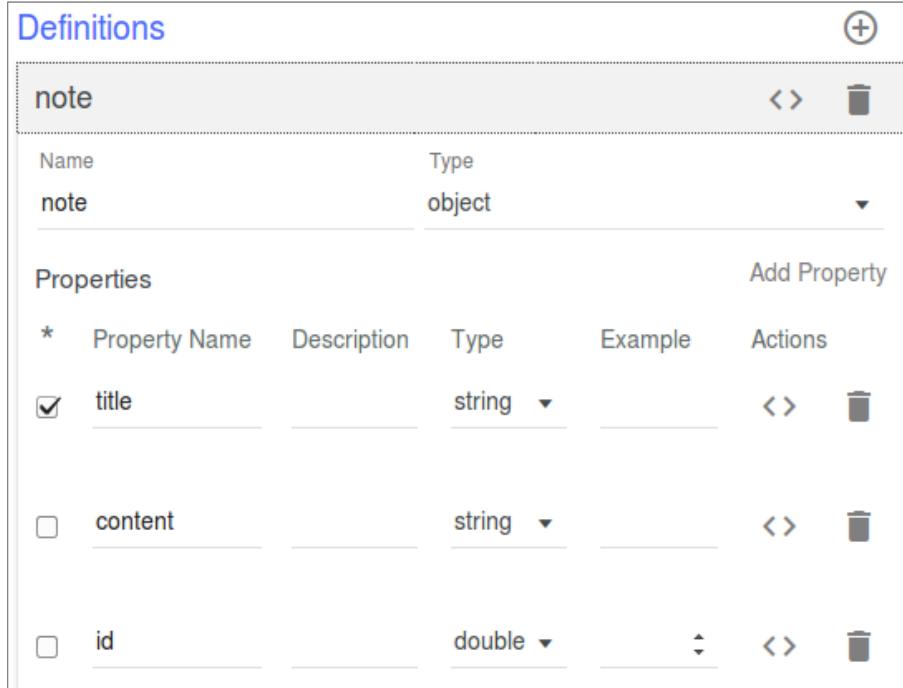
© Copyright IBM Corporation 2016

Figure 3-16. Reference schema definitions from operations

In this sample **POST** operation, the body in the parameters area references a schema definition named **note**.

Schema definitions are defined under the Definitions area in the API Designer design view.

## Schema definitions in the API Designer



The screenshot shows the 'Definitions' pane in the IBM API Designer. A schema named 'note' is selected. The schema details are as follows:

Name	Type
note	object

The 'Properties' section lists the fields defined in the schema:

*	Property Name	Description	Type	Example	Actions
<input checked="" type="checkbox"/>	title		string		<> 
<input type="checkbox"/>	content		string		<> 
<input type="checkbox"/>	id		double		<> 

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-17. Schema definitions in the API Designer

The definition for the **note** schema is shown in the visual pane of the API Designer editor.

## Sample schema definition in YAML format

```
definitions:  
  note:  
    properties:  
      title:  
        type: string  
      content:  
        type: string  
      id:  
        type: number  
        format: double  
    required:  
      - title  
  additionalProperties: false
```

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-18. Sample schema definition in YAML format

The sample shows the **note** schema from the hello-world application that is used in the exercises.

The schema defines the properties, data types, and data formats for the data that is passed in the REST call.

## Reference the schema definition in the GET operation

```

get:
  tags:
    - note
  summary: Find a model instance by id from the data source.
  operationId: note.findById
  parameters:
    - name: id
      in: path
      description: Model id
      required: true
      type: string
      format: JSON
  responses:
    '200':
      description: Request was successful
      schema:
        $ref: '#/definitions/note'
  deprecated: false

```

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

*Figure 3-19. Reference the schema definition in the GET operation*

The sample shows the partial GET operation that references the **note** schema in YAML source code.

In the body of the response message, the data is passed in the JSON format and the data is validated to correspond with the schema definition.

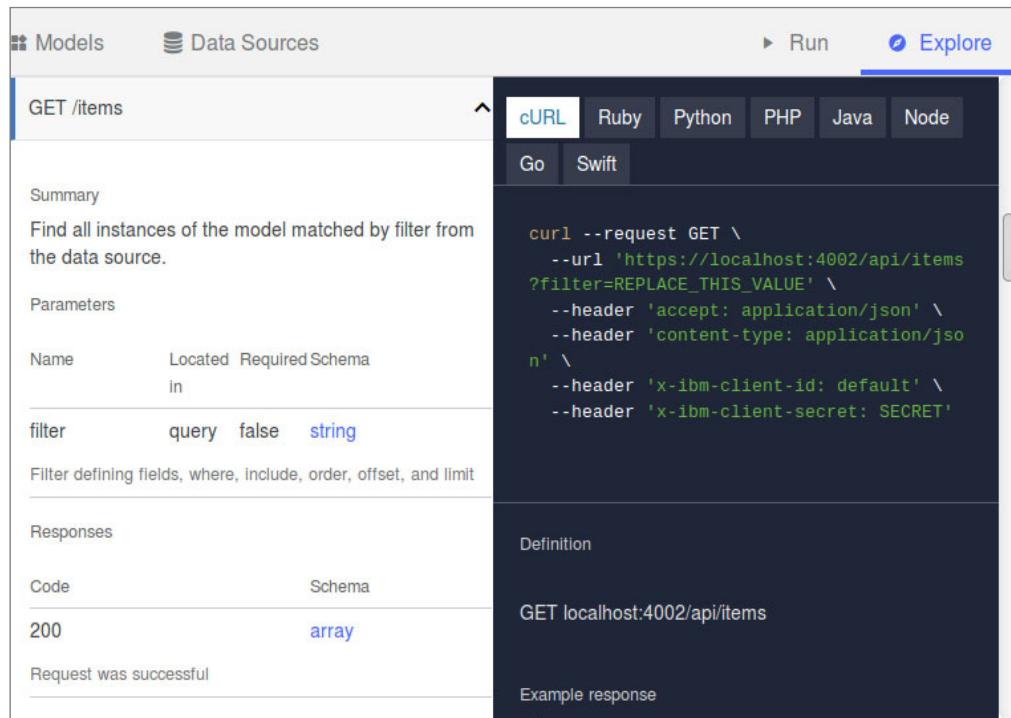
Operations that use the **note** schema can refer to schema definition with the YAML statements:  
schema:

\$ref: '#/definitions/note'

# IBM Training



## API Designer test feature



The screenshot shows the IBM API Designer interface. On the left, there's a sidebar with 'Models' and 'Data Sources' buttons, and a main area for the 'GET /items' operation. The operation summary states: 'Find all instances of the model matched by filter from the data source.' It lists a parameter 'filter' (query, false, string) and a response '200' (array). The response is described as 'Request was successful'. On the right, there's a 'Explore' tab with sub-tabs for 'cURL', 'Ruby', 'Python', 'PHP', 'Java', and 'Node'. Below these tabs, there are buttons for 'Go' and 'Swift'. A large text area contains cURL command examples. Below the command examples, there's a 'Definition' section with the URL 'GET localhost:4002/api/items' and an 'Example response' section.

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

Figure 3-20. API Designer test feature

The API Designer test feature lets you test the API locally on the workstation.

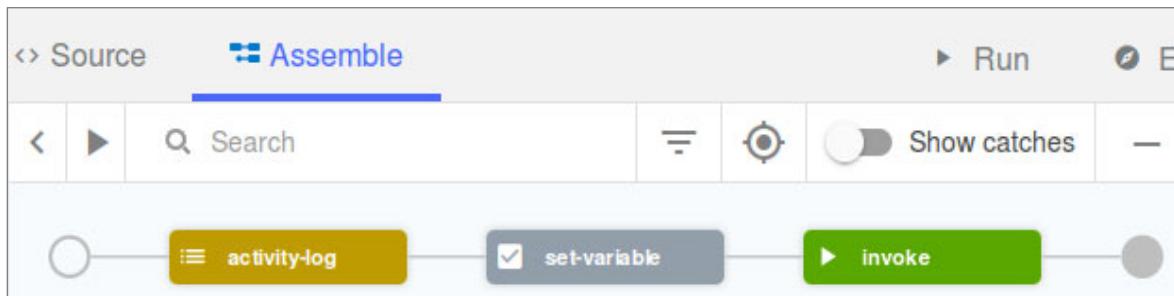
The test feature supplies sample code in multiple programming languages that lets you call the REST operations and view the results.

Scroll down in the test window until you see the Invoke button.

Selecting the Invoke button lets you test the operation and view the responses.

## Assemble view in API Designer

- Allows API developers to configure aspects of message flow
  - Log elements of the request or response message
  - Set variables
  - Perform conditional logic and message flow
  - Invoke policies



Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-21. Assemble view in API Designer*

In the assemble view, you can configure aspects of the request and response message flows when an API operation is called.

An assembly is formed of components that are applied to calls to and responses from operations in your API. Components can be either policies or logic constructs.

Policies are the building blocks of assembly flows, and they provide the means to configure capability, such as security, logging, routing of requests to target services, and the transformation of data from one format to another. Policies can be configured in the context of an API or in the context of a plan.

Logic constructs can be used to control the flow of messages. Logic constructs include conditional processing (if), operation switches (case statements), and throw (error).

## Assembly extensions in Swagger source

```
assembly:  
  execute:  
    - activity-log:  
        title: activity-log  
        content: payload  
        error-content: payload  
    - set-variable:  
        title: set-variable  
        actions:  
          - set: message.headers.Host  
            value: $(app-id)  
    - invoke:  
        target-url: $(app-server)$(request.path)  
        output: message
```

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

*Figure 3-22. Assembly extensions in Swagger source*

The page shows an example of assembly extensions in YAML source code that are supported in the API Connect product.

Use the assembly extension to describe the application of policies and logic to an API operation.

The assembly source is generated when you create an assembly with the visual editor from the Assemble tab of the API Designer that was displayed on the previous page.

## Unit summary

- Explain the concept of an API definition
- Explain the role of the Swagger specification in defining REST API interfaces
- Describe the structure of an API definition
- Design a REST API interface with the API Designer
- Explain the purpose of the assembly editor in the API Designer
- Create a REST API definition that invokes an existing system API

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-23. Unit summary

## Review questions

1. True or False: Swagger provides an intuitive user interface that helps developers to easily understand RESTful services.
  
2. Which of these statements is true about YAML:
  - A. Structure is defined by indentation
  - B. Easy for humans to read and edit
  - C. Easy for applications to parse and generate
  - D. All the above.
  
3. True or False: You can apply built-in policies by using the Assembly editor or by adding them in YAML source.



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-24. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: Swagger provides an intuitive user interface that helps developers to easily understand RESTful services.

The answer is True.



2. What is the first thing to check ...

- A. Structure is defined by indentation
- B. Easy for humans to read and edit
- C. Easy for applications to parse and generate
- D. All the above.

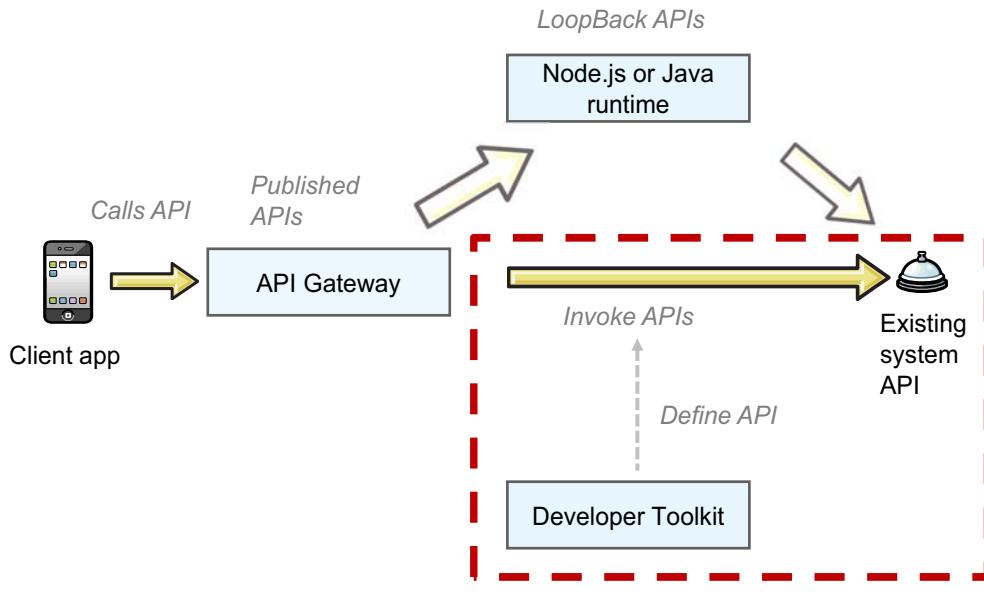
The answer is D.

3. True or False: You can apply built-in policies by using the Assembly editor or by adding them in YAML source.

The answer is True.

## Instructor demonstration

- In this scenario, you create a REST API definition to **invoke** an existing system API with the API Designer



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-26. Instructor demonstration

The scenario captures the process for defining an invoke API that calls an existing system API with API Connect.

LoopBack does not participate in this scenario.

## Instructor demonstration steps

- In this scenario, you create a REST API definition to invoke an existing system API with the API Designer.



1. Start the API Designer application
2. Define an API
3. Review the API definition
4. Define a path for the API
5. Create the address schema definition for the API
6. Create the branch schema definition for the API
7. Define the GET operation for the API
8. Configure the assembly to call an existing API
9. Review the source for the API definition
10. Start the application
11. Test the API operation from the Assemble tab

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

Figure 3-27. Instructor demonstration steps

The steps assume that you have an API Connect toolkit installed and configured on your local workstation or on the course image.

You must also be able to access the existing system API at  
<https://apim-services.mybluemix.net/banka/v1/branches>.

The demonstration goes through the steps in "Creating an invoke REST API definition" from the IBM API Connect Knowledge Center found at:

[http://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.toolkit.doc/tutorial\\_api\\_onprem\\_apiproxy.html](http://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/tutorial_api_onprem_apiproxy.html)

## Step 1: Start the API Designer application

1. Make a directory and subdirectory named **projects/branches**  
Then, change directory to the newly created directory

```
$ cd projects/branches
```

2. Start the API Designer application.

```
$ apic edit
```

3. Sign in to the Designer with your Bluemix account.



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-28. Step 1: Start the API Designer application

In the first step, open the terminal on Linux and Mac OS operating systems, or the command prompt on Windows operating systems.

If running the demonstration on the course image, create a **projects** directory, then create a subdirectory named **branches**. Change directory to the empty **projects/branches** directory.

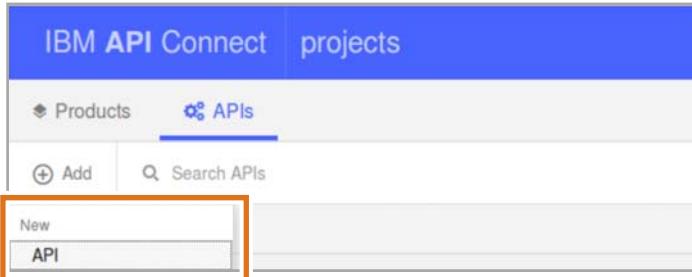
Type the command **apic edit** in the terminal.

If the API Designer is installed on your own workstation, sign on to the Designer with your Bluemix account.

The API Designer opens in the browser.

## Step 2: Define an API

1. In APIs tab, click the + Add icon. Then, select New API



2. In the Add an API wizard, create an API named branches
3. Change the base path to /branches
4. Click Next
5. Select Do not add to a product
6. Click Add

[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

Figure 3-29. Step 2: Define an API

In the APIs tab, click the + Add icon. Then, select API in the New dialog.

Create an API named branches.

Change the base path to /branches.

Leave the remaining fields as their default values.

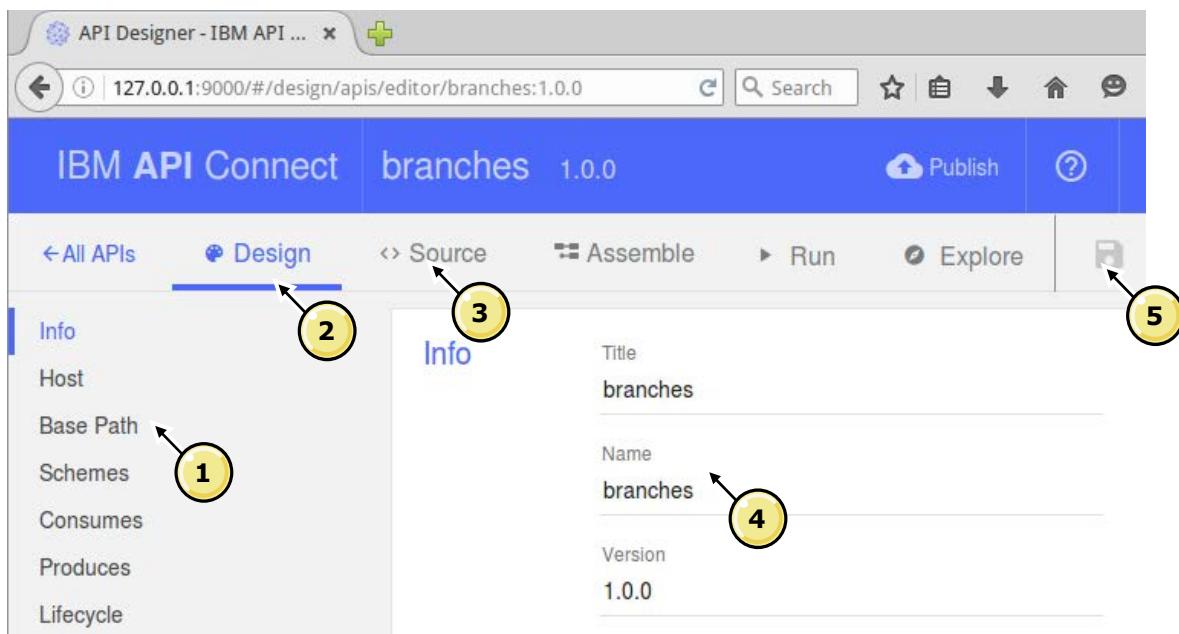
Click Next.

Leave the default to not add a product.

Click Add.

## Step 3: Review the API definition

- The wizard creates a Swagger file that represents your draft API. Use the **API Editor** to review and define your API details.



[Define an API with the API Designer](#)

© Copyright IBM Corporation 2016

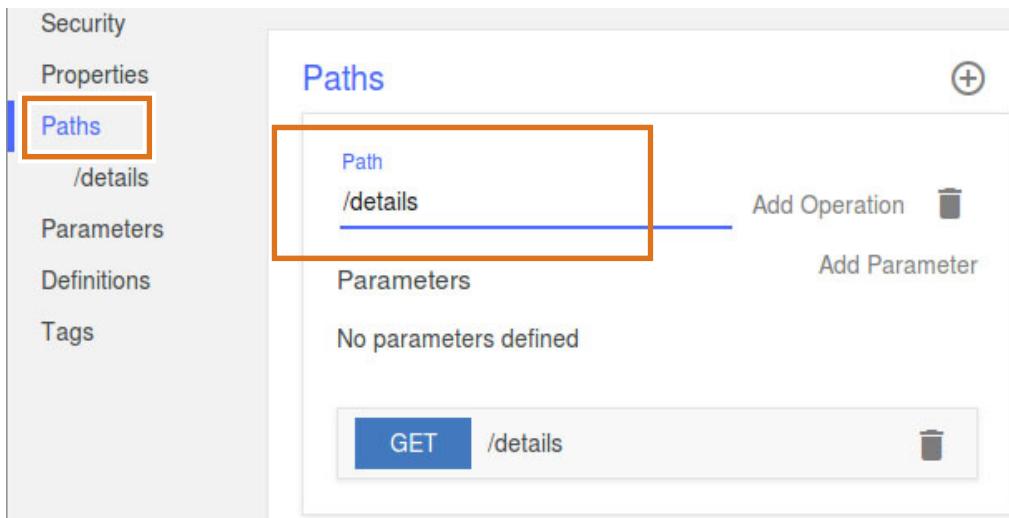
Figure 3-30. Step 3: Review the API definition

The **API Designer** is a web application form that edits your application's API definition. The Designer is part of the **API Editor** web application.

- The **API definition** specifies the base path and other definitions in your API. API Connect uses the Swagger V2.0 document format for the API Definition. In this leftmost column, the API Designer view lists the API definition categories.
- To open the API Designer, you typed apic edit. Then, you added a new API. By default, the **Design** tab is .
- You can also edit the source version of the API definition in a built-in text editor, in the **Swagger YAML** (Yet another markup language) format.
- In this example, you specified the title, name, version number, and base path for your API.
- When you make any changes to the file, select **Save** to commit your changes.

## Step 4: Define a path for the API

1. Select **Paths** from the categories
2. Click + in the Paths area to add a path
3. Replace the default path segment with /details



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-31. Step 4: Define a path for the API

Select Paths from the categories in the navigation panel. Click the icon in the Paths area to add a path.

Replace the default path segment with /details.

When an operation is called, this path segment is appended to the URL for the API.

## Step 5: Create the address schema definition for the API

1. In the Definitions area, click the icon to add a definition
2. Change the Name field to address
3. Create new properties by clicking Add Property. Then, define:
  - street1, street2, city, state, zip\_code all with type string

Name	Type
address	object

**Properties**

* Property Name	Description	Type	Example	Actions
street1		string		<>
street2		string		<>
city		string		<>
state		string		<>
zip_code		string		<>

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-32. Step 5: Create the address schema definition for the API

Scroll to the Definitions area, then click the icon to add a definition.

Type **address** in the Name field.

Create new properties by clicking Add Property.

Add the properties street1, street2, city, state, and zip\_code all with the string data type.

## Step 6: Create the branch schema definition for the API

1. In the Definitions area, click the icon to add a definition
2. Change the Name field to **branch**
3. Create new properties by clicking Add Property. Then, define properties: **id**, **type**, **address**

Properties						Add Property
*	Property Name	Description	Type	Example	Actions	
<input type="checkbox"/>	id		string			
<input type="checkbox"/>	type		string			
<input type="checkbox"/>	address		address			

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-33. Step 6: Create the branch schema definition for the API

Scroll to the Definitions area, then click the icon to add a definition.

Type **branch** in the Name field.

Create new properties by clicking Add Property.

Add the properties **id**, **type**, and **address**. The properties **id** and **type** must be assigned the **string** data type.

The property named **address** must be assigned the **address** data type.

## Step 7: Define the GET operation for the API

1. Click **GET /details** to in the path area to edit the operation
2. In the Summary field, type **Branch details**
3. Select **branch** in the Schema drop-down
4. **Save** the API definition

The screenshot shows the API Designer interface with the following details:

- Path:** GET /details
- Summary:** Branch details
- Operation ID:** (empty)
- Parameters:** No parameters defined
- Responses:**

Status Code	Description	Schema
200	200 OK	branch ▾
- Security:**
  - Enable security definitions
  - Use API security definitions
  - clientID (API Key)

Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-34. Step 7: Define the GET operation for the API

By default, a single GET operation is already in your path.

Click the operation to expand its details.

Type **Branch details** in the Summary field.

Select **branch** from the Schema drop down.

Save the changes.

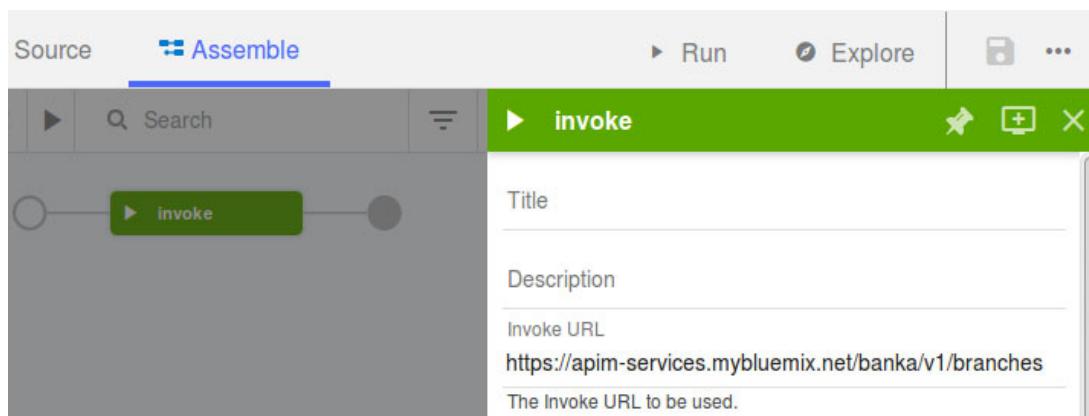
You have now created the API definition.

Next, you use an assembly to invoke an existing API endpoint.

The assembly is used by the test tool or the gateway at runtime to call the existing API.

## Step 8: Configure the assembly to call an existing API

1. Click the **Assemble** tab in API Designer
2. Click the **invoke** policy in the flow to open the Properties dialog
3. In the Properties for invoke, type the endpoint in the **Invoke URL** field:  
`https://apim-services.mybluemix.net/banka/v1/branches`
4. Save the changes



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-35. Step 8: Configure the assembly to call an existing API

Click the **Assemble** tab in API Designer.

A default invoke policy is already displayed in the flow area.

Click **invoke** to open the Properties dialog.

In the Invoke URL field, type `https://apim-services.mybluemix.net/banka/v1/branches`

Close the Properties dialog.

Save the changes.



## Step 9: Review the source for the API definition

- Click the **Source** tab in API Designer to view the Swagger YAML file

```

1  swagger: '2.0'
2  info:
3    version: 1.0.0
4    title: branches
5    x-ibm-name: branches
6    host: ${catalog.host}
7    basePath: /branches
8    paths:
9      /details:
10     get:
11       responses:
12         '200':
13           description: 200 OK
14           schema:
15             $ref: '#/definitions/branch'
16           summary: Branch details
17   securityDefinitions:
18     clientID:
19       description: ''
20       in: header
21       name: X-IBM-Client-Id
22       type: apiKey
23   security:
24     - clientID: []
25   x-ibm-configuration:
26     assembly:
27     execute:
28       - invoke:
29         target-url: 'https://apim-services.mybluemix.net/banka/v1/branches'
30   definitions:
31     branch:
32       properties:
33         id:
34           type: string
35         type:
36           type: string
37         address:
38           type: string
39           $ref: '#/definitions/address'

```

Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-36. Step 9: Review the source for the API definition*

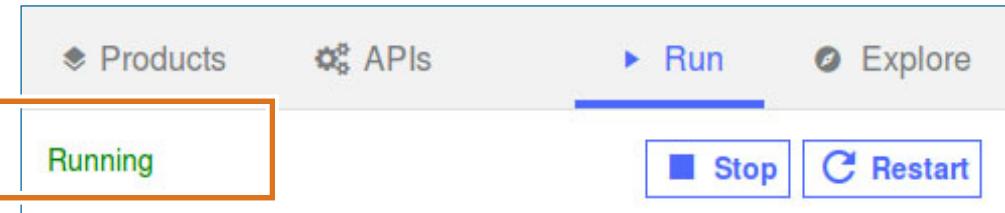
Click the **Source** tab in API Designer to view the Swagger definition for your API.

All the configuration that you have performed is included in this definition, either as part of the standard Swagger schema, or as part of the x-ibm-configuration extension.

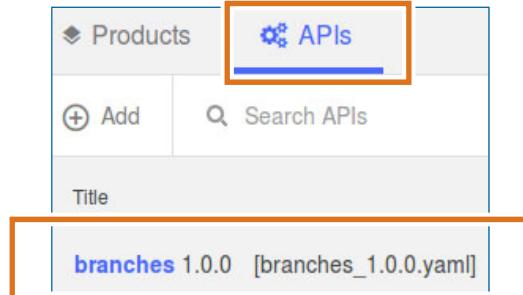


## Step 10: Start the application

1. Start the application
2. Click **Run**, then **Start**
  - When you see the Running message, the application is started



3. Click the **APIs** tab.
4. Select the **branches** API



Define an API with the API Designer

© Copyright IBM Corporation 2016

*Figure 3-37. Step 10: Start the application*

Start the application.

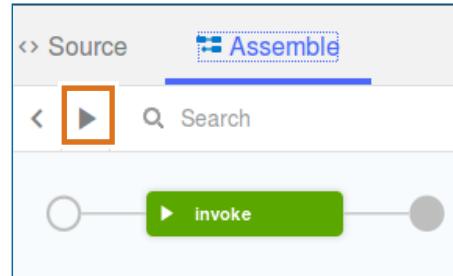
In the API Designer, click **Run**. Then, click **Start**.

Wait until the Running message is displayed. Then, click the **APIs** tab.

Click the **branches** link.

## Step 11: Test the API operation from the Assemble tab

1. Click the **Assemble** tab
2. Click the Test icon



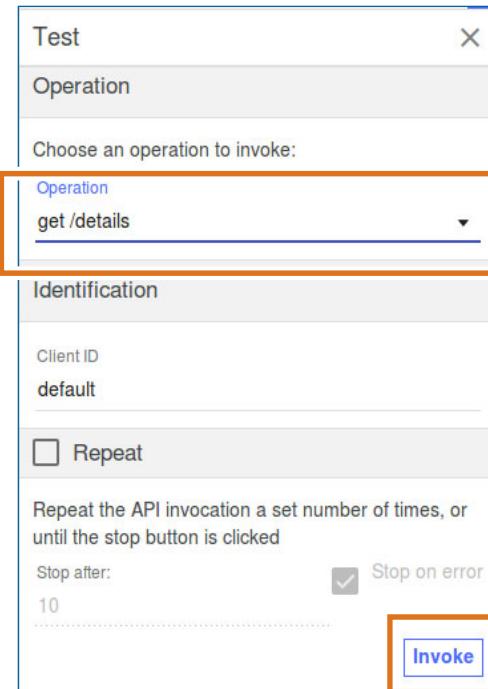
Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-38. Step 11: Test the API operation from the Assemble tab

## Step 12: Select the API operation

1. In the **Operation** section, use the drop-down to select the operation
2. Click **Invoke**



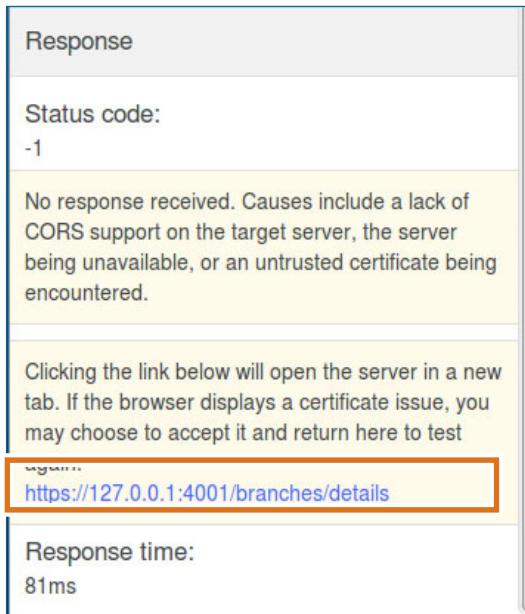
Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-39. Step 12: Select the API operation

## Step 13: Enable CORS in the browser (1 of 2)

1. If a page is displayed that indicates a lack of CORS support, click the hyperlink



The screenshot shows a 'Response' panel from a web application. At the top, it says 'Status code: -1'. Below that, a message states: 'No response received. Causes include a lack of CORS support on the target server, the server being unavailable, or an untrusted certificate being encountered.' A note below the message says: 'Clicking the link below will open the server in a new tab. If the browser displays a certificate issue, you may choose to accept it and return here to test.' A blue hyperlink 'https://127.0.0.1:4001/branches/details' is highlighted with a red border. At the bottom, it says 'Response time: 81ms'.

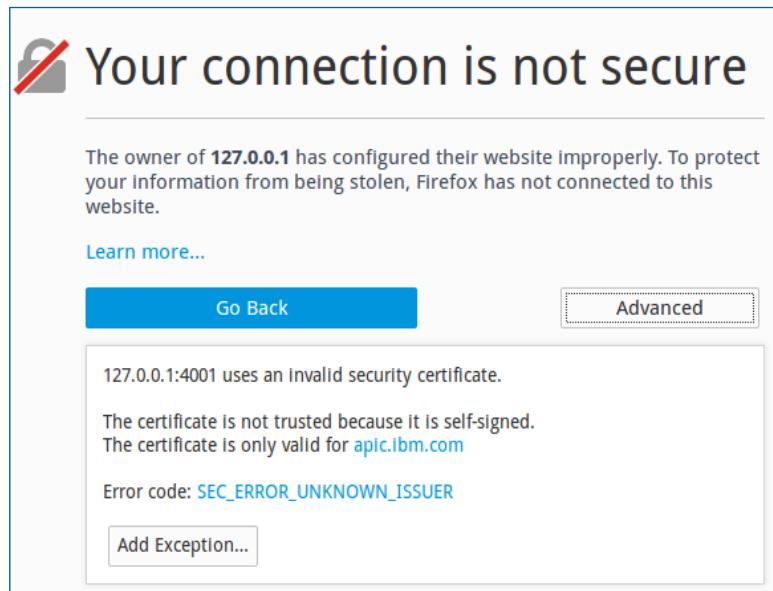
Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-40. Step 13: Enable CORS in the browser (1 of 2)

## Step 13: Enable CORS in the browser (1 of 2)

1. Click **Advanced** in the browser. Then, click **Add Exception**.



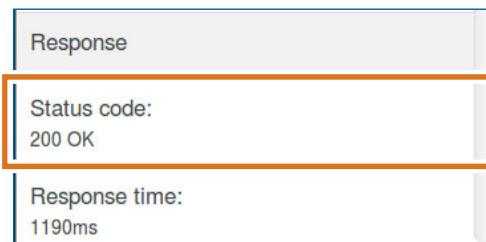
Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-41. Step 13: Enable CORS in the browser (1 of 2)

## Step 14: Rerun test

1. Click **Invoke** icon in the Operation section.
2. The successful response status code 200 is returned



Define an API with the API Designer

© Copyright IBM Corporation 2016

Figure 3-42. Step 14: Rerun test

---

# Unit 4. Implement an API as a Node.js Loopback application

## Estimated time

01:15

## Overview

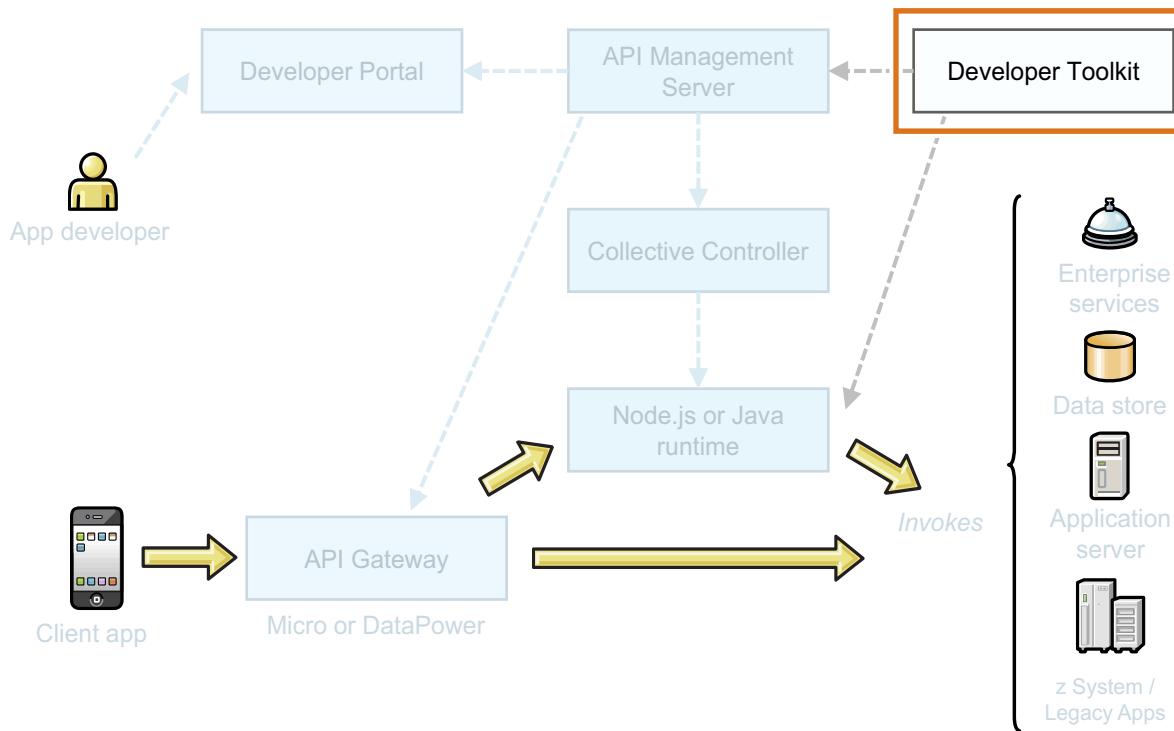
In this unit, you learn how to model and implement a REST API as a LoopBack application. With the LoopBack framework, you can quickly implement REST APIs from data models in a Node.js application. Learn how to define models, properties, and relationships in a LoopBack application. Configure data sources to access and persist model data. Test your API implementation in the API Explorer.

## Unit objectives

- Explain the concept of a LoopBack model, property, and relationship
- Explain the role of a LoopBack data source
- Explain how to configure a LoopBack connector
- Identify the role of database and non-database connectors



## API Connect: Topology view



Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-2. API Connect: Topology view

The topology view displays how the six components of IBM API Connect relate to one another.

This unit focuses on the LoopBack generation features in the `apic` command-line utility. The utility is part of the API Connect Developer Toolkit.

## 4.1. LoopBack framework in API Connect

## LoopBack framework in API Connect

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

*Figure 4-3. LoopBack framework in API Connect*

## Topics

### LoopBack framework in API Connect

- LoopBack models, properties, and relationships
- LoopBack data sources and connectors

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-4. Topics

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

- LoopBack is an open source Node.js application framework for REST APIs.
  - See <http://loopback.io> for documentation and source code on the LoopBack open source project.
- 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 pre-defined REST API with a full set of create, retrieve, update, and delete operations.

## Example: Hello-world LoopBack application

- In the **hello-world** LoopBack sample application, the **note** model stores a greeting message.
  - The message consists of two fields: **title**, and **content**.
  - **The model defines a static method that is named greet.** The greet method returns the input query parameter that it receives.
- 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.
- The LoopBack framework also maps the **greet** Node.js function to an API operation, **GET /notes/greet**.
  - When you call **GET /notes/greet?msg="IBM"**, the API operation returns `{ "greeting" : "Hello IBM" }` as a response message.

[Implement an API as a Node.js Loopback application](#)

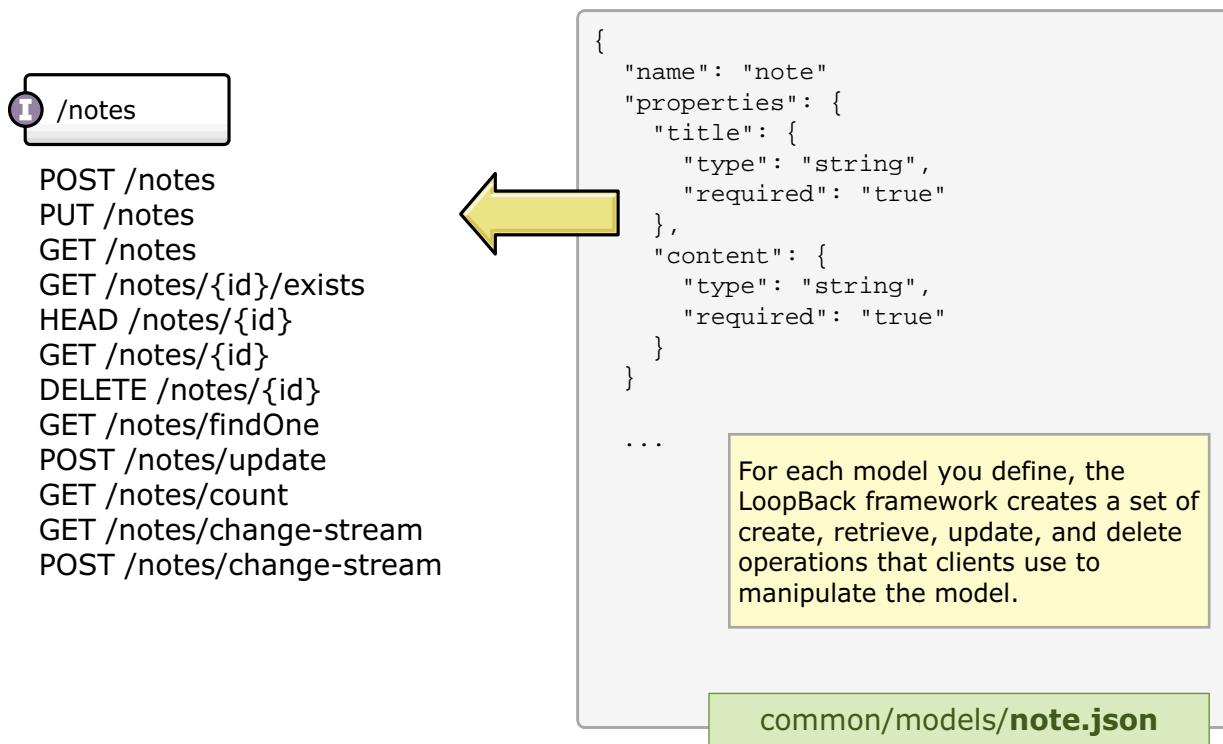
© Copyright IBM Corporation 2016

Figure 4-6. Example: Hello-world LoopBack application

As a static method, the **greet** function does not use the data from the **title** and **content** properties from the **notes** model.

To create a copy of the hello-world LoopBack application, run *apic loopback* and select the *hello-world* application as a starting point. You examined the hello-world application in the first exercise of this course.

## Example: Hello-world note model (1 of 2)



Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-7. Example: Hello-world note model (1 of 2)

In the first exercise, you generate the Hello-World 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** – Returns status code 200 OK if a note with the specified identifier exists.

**GET /notes/{id}** – Retrieve the title and content for the specified note.

**HEAD /notes/{id}** – Returns 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: there is no assumed ordering in the notes.

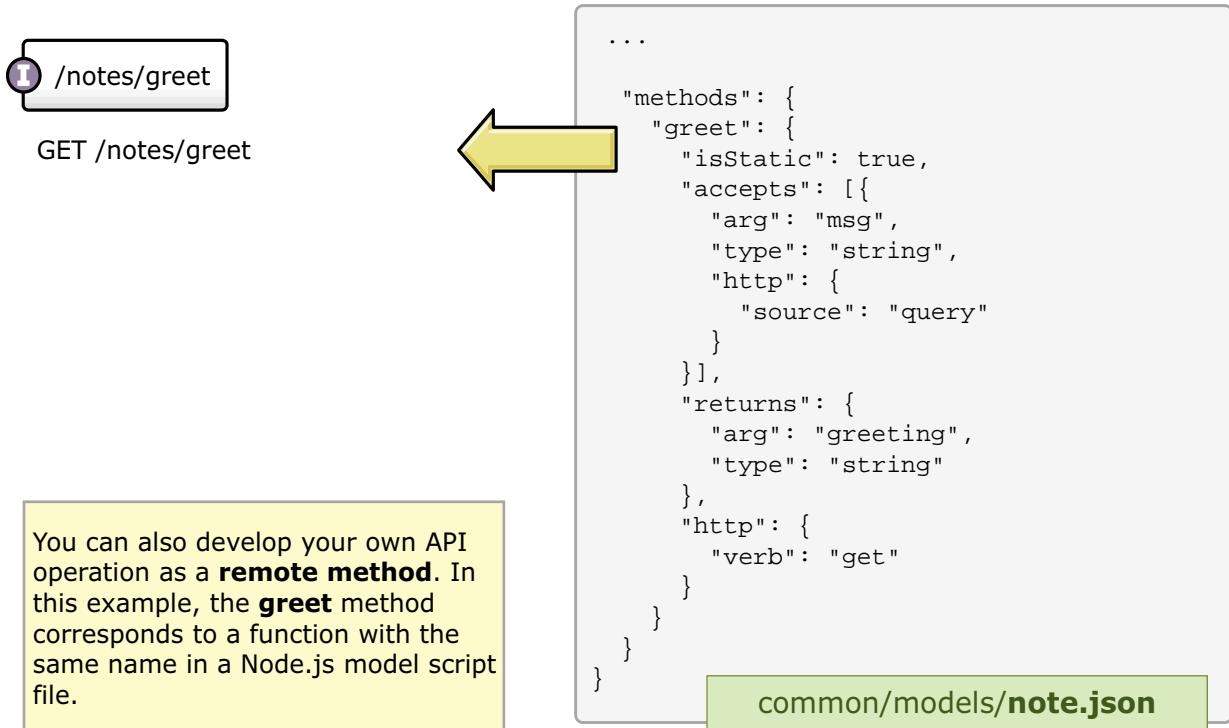
**POST /notes/update** – Update an existing note. This operation is the same as **PUT /notes**.

GET /notes/count – Returns the total number of notes.

GET /notes/change-stream – Returns 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 note model (2 of 2)



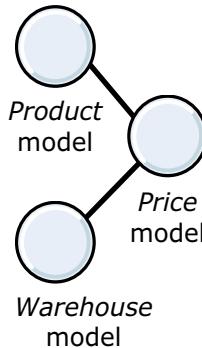
Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-8. Example: Hello-world note model (2 of 2)

## 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.



Implement an API as a Node.js Loopback application

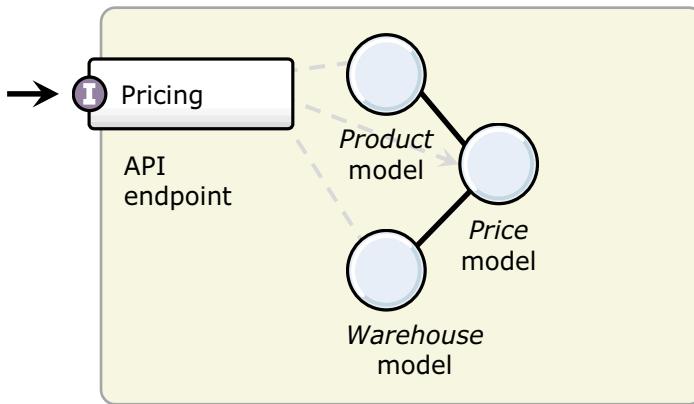
© Copyright IBM Corporation 2016

Figure 4-9. 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.



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

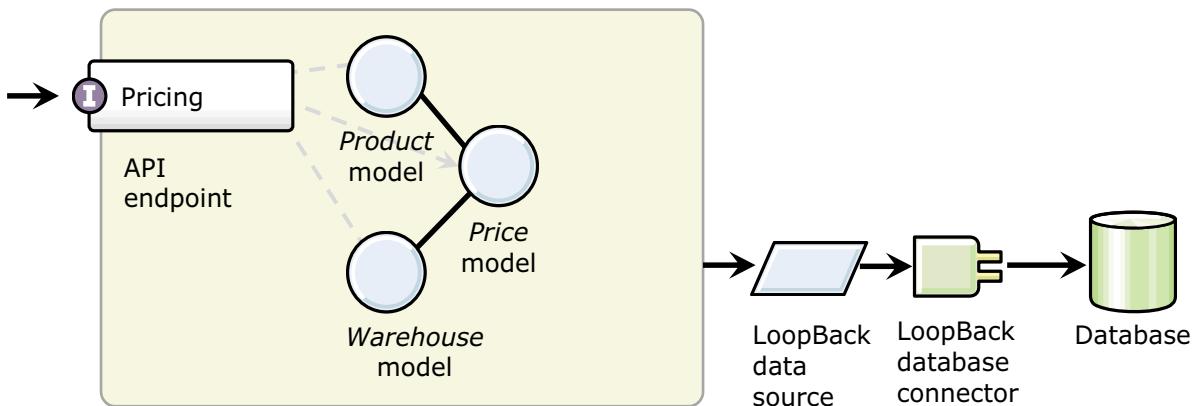
Figure 4-10. LoopBack framework: API mapped to models

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

As a recommended 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**.



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

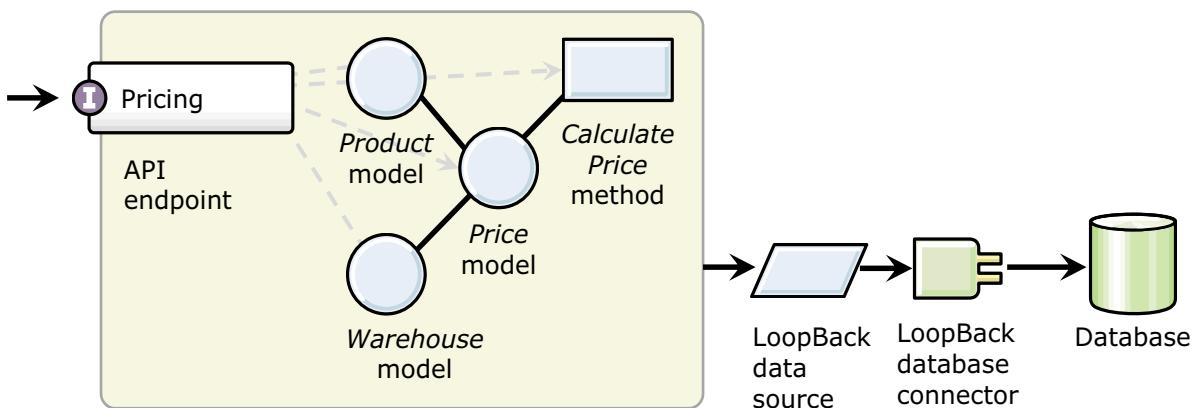
Figure 4-11. LoopBack framework: data persistence with connectors

A **LoopBack connector** is a Node module that connects model objects to sources of data outside of your LoopBack application. There are two categories of LoopBack connectors: 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.

There are three steps in configuring a database connector. First, you install a **LoopBack connector** for your type of database. Second, you define the connection information in a **LoopBack data source**. Last, 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.



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

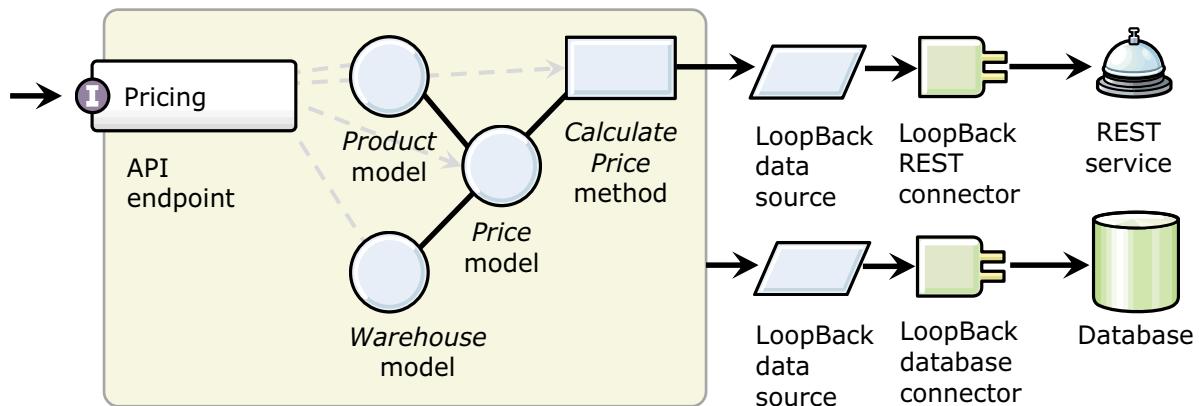
Figure 4-12. LoopBack framework: remote methods and hooks

The *calculate 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 *calculate 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.

Remote hooks are event handlers that execute 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.



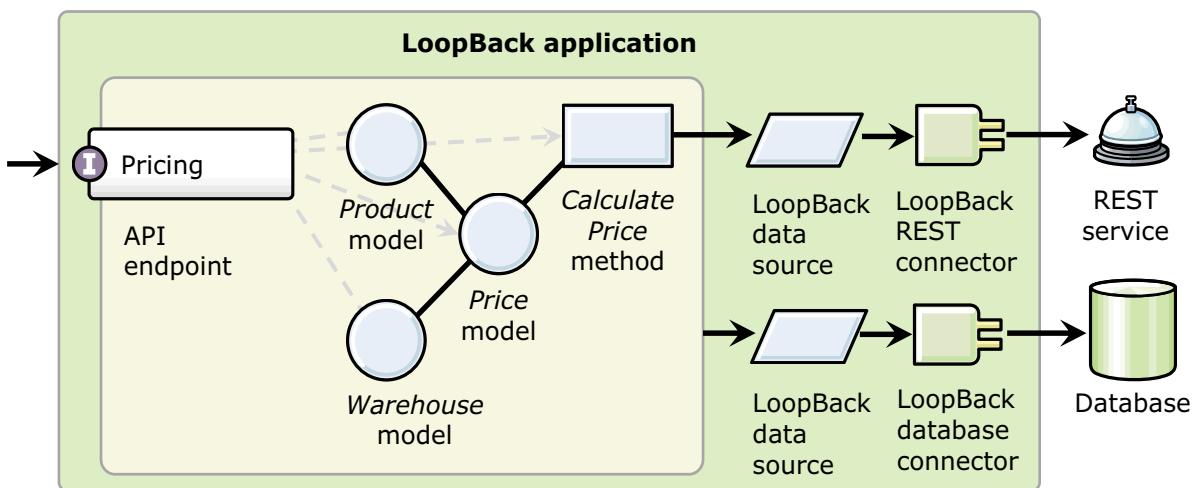
Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-13. 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.



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-14. LoopBack framework: packaging

A LoopBack application is a Node.js application built around the structure of the LoopBack framework. The LoopBack framework maps a set of models to a pre-configured set of API operations. In this scheme, you define the structure of your models through configuration files. You implement free-form API operations as Node.js functions known as remote methods. You write code that intercepts API operations in Node.js functions 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 interaction APIs**, hosted on a server that the API Gateway can access.
- A **LoopBack** application is one choice for **implementing interaction 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.

[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

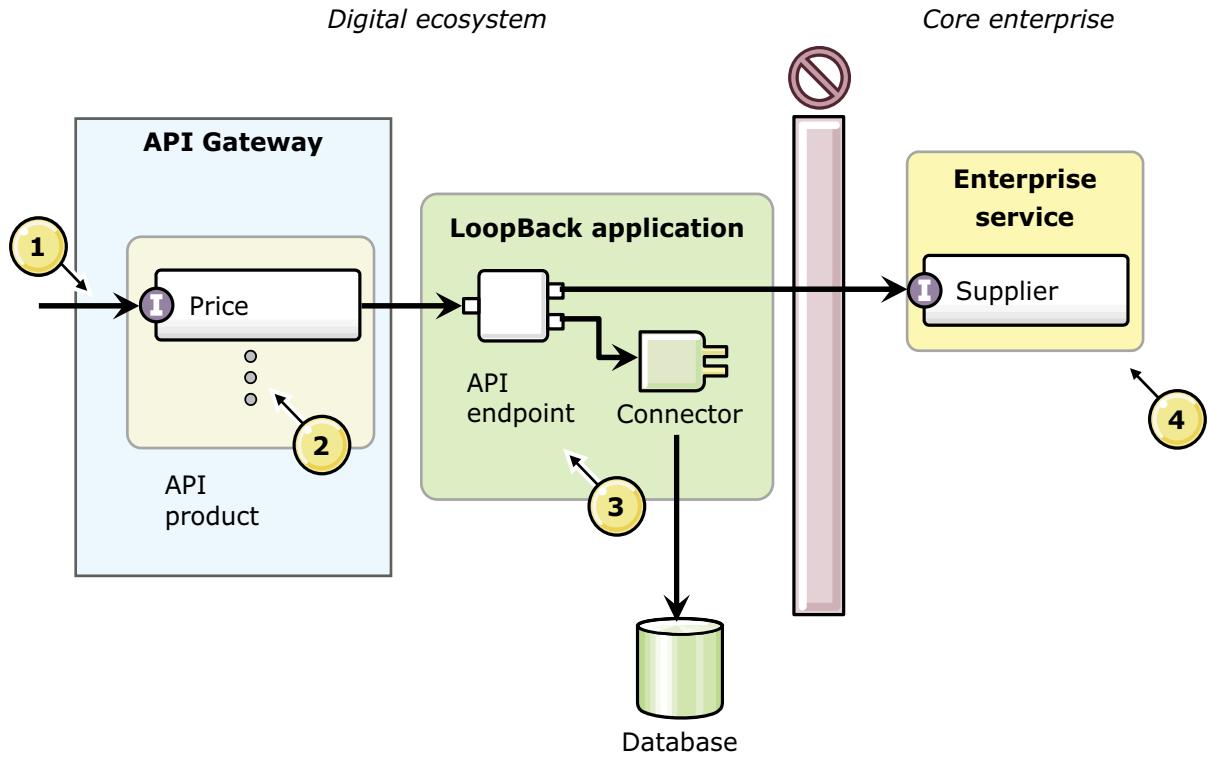
*Figure 4-15. The role of LoopBack in API Connect*

Specifically, the publish operation occurs in several stages. As an API developer, you define the interface for an API in an API definition. You package an API definition in a Product. From the API Designer, you publish both the Product and API definition to the API Manager.

The API Manager publishes the API definition files to the API Gateway.

You learn about the lifecycle of APIs and the publish process in a later unit.

## LoopBack applications in an API Connect solution



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-16. 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, or on IBM Bluemix.
4. The interaction API that you implement in LoopBack can call other interaction 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 quite different from the endpoint that the gateway calls the Loopback application.

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

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

## Instructor demonstration

- 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.

[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

*Figure 4-18. Next steps after implementing the API*

You deploy a LoopBack application in the same manner as a Node.js application. For an example of LoopBack application deployment, see:

## 4.2. LoopBack models, properties, and relationships

## LoopBack models, properties, and relationships

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

*Figure 4-19. LoopBack models, properties, and relationships*

## Topics

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

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-20. 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.**

## 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**.
- There are two starting points for your LoopBack application:
  - An **empty-server API** does not define any configured models or data sources.
  - The **hello-world API** defines the **note** model and saves its information into an in-memory data source.

## Example: Create a LoopBack application

\$ apic loopback ← 1

Let's create a LoopBack application! ← 2

? What's the name of your application? pricing ← 3

? Enter name of the directory to contain the project: pricing ← 4

? 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) ← 5

hello-world (A project containing a basic working example, including a memory database)

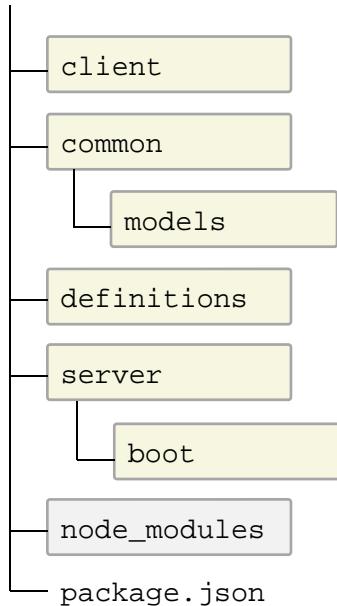
Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-23. Example: Create a LoopBack application

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

## 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** store 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.

[Implement an API as a Node.js Loopback application](#)

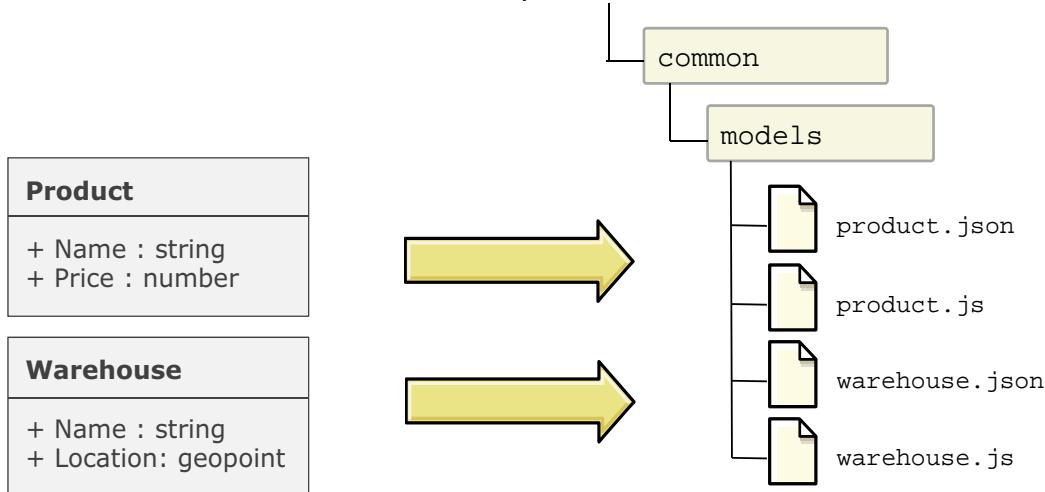
© Copyright IBM Corporation 2016

Figure 4-24. 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.



Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-25. 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 **warehouse** model has a **name** in string format, and a **location** as a geopoint. The geopoint holds latitude and longitude co-ordinates; it is a data type specific to the LoopBack framework.

The **model definition** file saves the characteristics of each model object. For the

## Example: Create a model and properties

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

```
? 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:
```

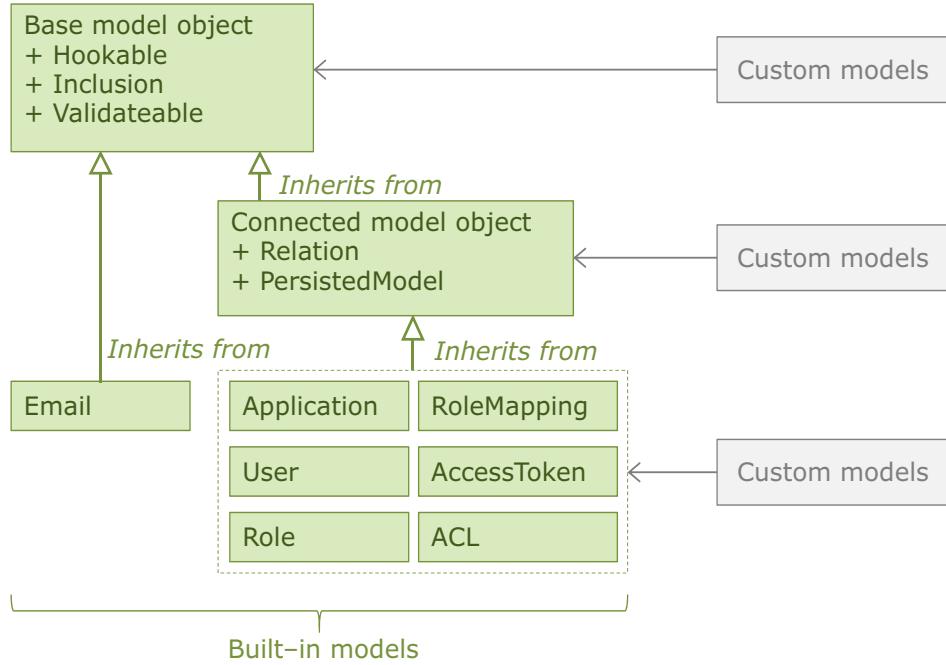
[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-26. 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 pre-configured 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 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



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-27. LoopBack model inheritance

When you create a model object in your LoopBack application, the utility asks you for 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.

## 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 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**

Figure 4-28. 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

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

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

common/models/**product.js**

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

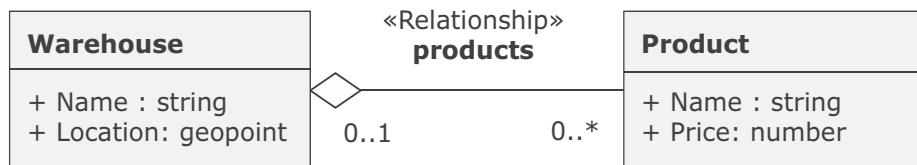
For example, the **product** model extends the **persisted model** base class. Therefore, you do not need to write code to persist model data to a data source.

Figure 4-29. 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 pre-defined 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 named **products** in the **warehouse** model.
  - You define the model relationship in `warehouse.json`.
  - Loopback adds a set of **search and query APIs** that retrieves product models that the warehouse owns.



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-30. How do you define a relationship between models?

In this example, you define a one-to-many relationship in the **warehouse** model. The warehouse has many products in its inventory.

To save this relationship information, you create a relationship with the `apic` command-line utility. The command adds a “has many” relationship in the **warehouse** model definition file, `warehouse.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 product to a warehouse with the `POST /warehouses/products/` API operation.

## Example: Create model relations

```
$ apic loopback:relation
```

? Select the model to create the relationship from:

- product**
- warehouse**

? Relation type:

- has many**
- belongs to
- has many and belongs to
- has one

? Choose a model to create a relationship with:

- product**
- warehouse
- (other)

? Enter the property name for the relation:

- products**

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-31. 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 **warehouse** 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 warehouse to the **product** 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 **warehouse has many products** in its inventory. The name of the relation is **products**.

## Example: Relations in the model definition file

```
{  
  "name": "warehouse",  
  ...  
  "relations": {  
    "products": {  
      "model": "product",  
      "type": "hasMany",  
      "foreignKey": "productId"  
    }  
  },  
  ...  
}
```

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 **products**. The warehouse model **has many** products. The way that you track the relationship is to store a collection of **productId** values.

common/models/**warehouse.json**

Figure 4-32. Example: Relations in the model definition file

## Model relation types

- For example, you create two models named **product** and **warehouse**.

Relation type	Example
Has many	A <b>warehouse</b> has many <b>products</b> .
Belongs to	A <b>product</b> belongs to exactly one <b>warehouse</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. Create a third model (a through-model) that maps <b>warehouses</b> to <b>products</b> .

- For a detailed explanation on relation types, see:  
<https://docs.strongloop.com/display/APIC/Creating+model+relations>

## 4.3. LoopBack data sources and connectors

## LoopBack data sources and connectors

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

*Figure 4-34. LoopBack data sources and connectors*

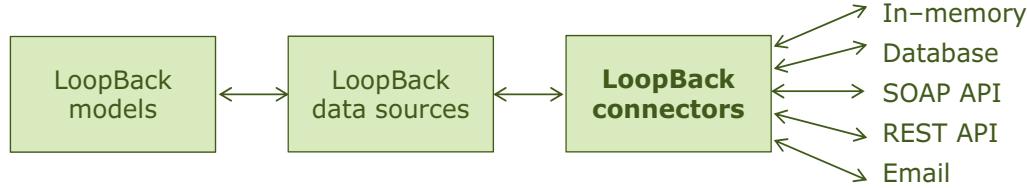
## Topics

- LoopBack framework in API Connect
- LoopBack models, properties, and relationships
-  LoopBack data sources and connectors

## 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.

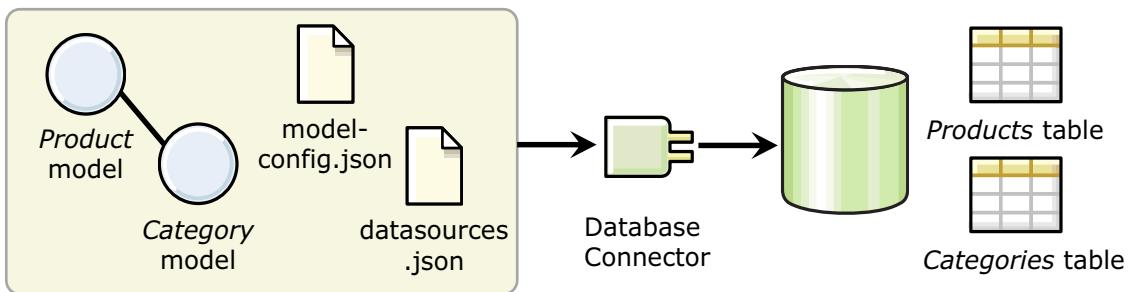
## What is a LoopBack connector?



- LoopBack connectors **connect models to external sources of data** through create, retrieve, update, and delete functions.
- LoopBack also generalizes other backend 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 access data sources through the `DataSource` and `PersistedModel` APIs.

## 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.



[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-38. 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
  - IBM DB2
  - Memory
  - MongoDB
  - MySQL
  - Oracle
  - PostgreSQL
  - Redis
  - SQL Server
- Review the documentation for an up-to-date list of database connectors:  
<https://docs.strongloop.com/display/public/LB/Database+connectors>

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-39. Database connectors

The list of LoopBack connectors are 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 ← 1
? What's the name of your application? memoryds ← 2
? 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) ← 3

Connector-specific configuration:
? window.localStorage key to use for persistence (browser only):
? Full path to file for persistence (server only):
  ./server/data/memory.json ← 4 ← 5
```

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-40. 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, as well 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 to blank.
5. In the second question, you have the option to 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 4-41. 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

```
{
  "_meta": {
    "sources": [
      "loopback/common/models",
      "loopback/server/models",
      "../common/models",
      "./models" ],
    "mixins": [
      "loopback/common/mixins",
      "loopback/server/mixins",
      "../common/mixins",
      "./mixins" ]
  },
  "product": {
    "dataSource": "memoryds",
    "public": true
  },
  "warehouse": {
    "dataSource": "memoryds",
    "public": true
  }
}
```

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

The **\_meta** section is used by the LoopBack framework. 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 **dataSource** declares which data source LoopBack uses to retrieve data. The **public** setting determines whether LoopBack adds API operations for the model object.

**server/model-config.json**

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-42. 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 co-ordinate 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

[Implement an API as a Node.js Loopback application](#)

© Copyright IBM Corporation 2016

Figure 4-43. Database transactions

The purpose of database transactions is to execute 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.

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 performs a **SELECT** statement. It is susceptible to phantom reads, and non-repeatable reads.

**Read uncommitted** is the lowest isolation level. It allows dirty reads, and it is susceptible to phantom reads and non-repeatable reads.

A *dirty read*, or an uncommitted read, allows a transaction to read data that another transaction modified, but not committed.

In a *non-repeatable read*, two queries within the same transaction returns different values for the same row. This phenomenon occurs if another transaction modifies a row before the database executes the second query.

The phenomenon of *phantom reads*, in which a transaction that executes two identical queries receives different collections of rows. This phenomenon occurs if another transaction inserts a row before the database executes the second query.

## Non-database connectors

- Beyond databases, LoopBack supports a set of connectors that implement specific ways to access remote systems.
  - Email connector
  - Push connector
  - Remote connector
  - REST connector
  - SOAP connector
  - Storage 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 on each non-database connector type, see:  
<https://docs.strongloop.com/display/public/LB/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.

## Unit summary

- Explain the concept of a LoopBack model, property, and relationship
- Explain the role of a LoopBack data source
- Explain how to configure a LoopBack connector
- Identify the role of database and non-database connectors

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

*Figure 4-45. Unit summary*

## Review questions

1. True or False: You must extend the persisted model base class in order to use a database connector.
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.



Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-46. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: You must extend the persisted model base class in order to use a database connector.  
The answer is True.
  
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.



## Exercise: Create a LoopBack application

### Lab 2

Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-48. Exercise: Create a LoopBack application

## Exercise objectives

- Create an API definition with the API Designer web application
- Generate an API definition with the *apic* command-line utility
- Configure the MySQL connector
- Configure the MongoDB connector
- Create LoopBack models and properties
- Define relationship between models
- Test an API with the API Explorer



Implement an API as a Node.js Loopback application

© Copyright IBM Corporation 2016

Figure 4-49. Exercise objectives

---

# Unit 5. Implement and publish LoopBack API applications

## Estimated time

01:00

## Overview

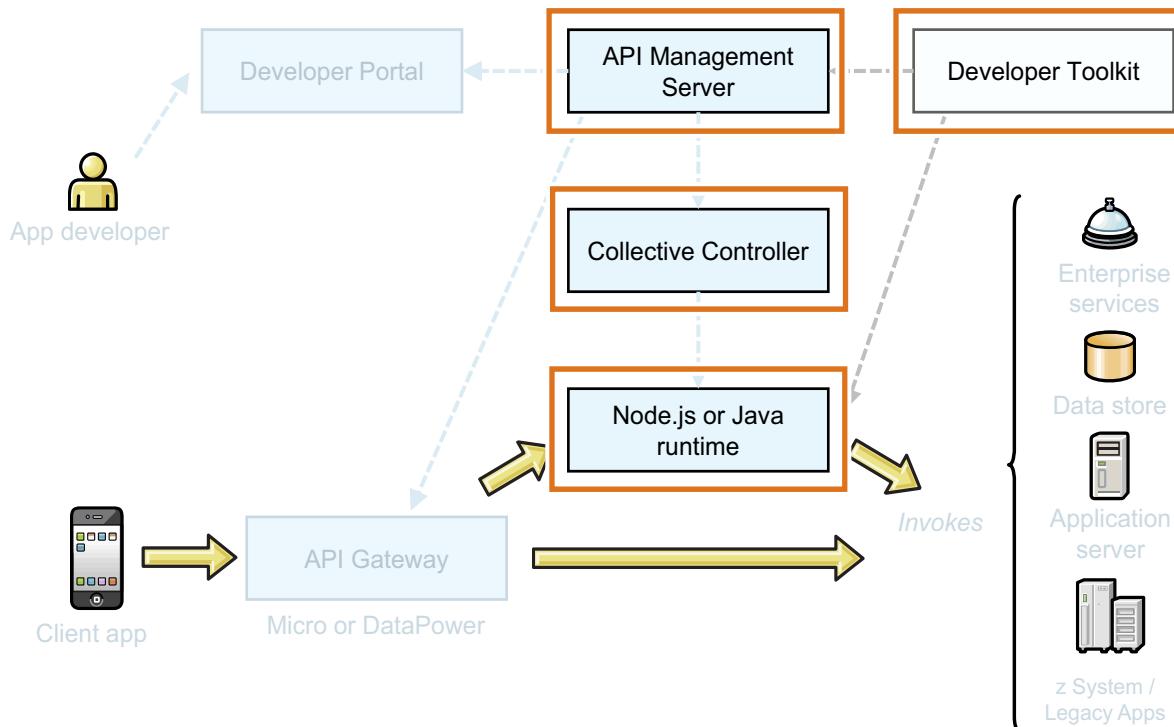
In this unit, you learn how to implement and publish LoopBack applications. You learn how to extend the data-centric model of LoopBack applications with remote methods. You also learn how to implement event-driven functions in the form of remote and operation hooks. Last, you learn how to deploy LoopBack API applications with Collectives in IBM API Connect architecture.

## Unit objectives

- Explain the purpose of a remote method
- Explain the purpose of a remote and operation hook
- Explain the role of Collectives in Node.js application deployment
- Explain how to deploy LoopBack API applications with a Collective Controller



## API Connect: Topology view



Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-2. API Connect: Topology view

The topology view displays how the six components of IBM API Connect relate to one another.

This unit focuses on the Collective Controller, API Management Server, and a host environment to run Node.js LoopBack applications.

## 5.1. LoopBack remote methods and hooks

## LoopBack remote methods and hooks

Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

*Figure 5-3. LoopBack remote methods and hooks*

## Topics

- ▶ LoopBack remote methods and hooks
  - Node.js application deployment with Collectives

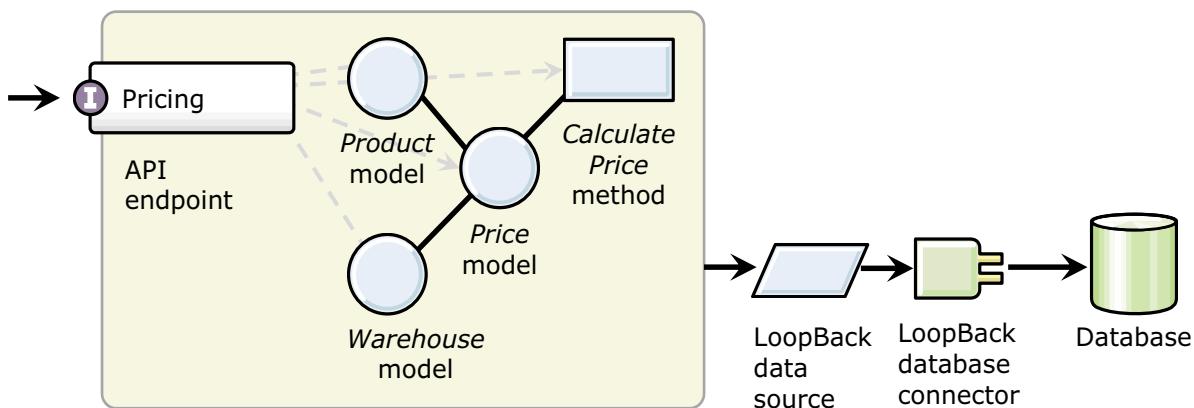
Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-4. 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.



Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-5. LoopBack framework: remote methods and hooks

The *calculate 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 *calculate 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.

Remote hooks are event handlers that execute 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.

## 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.

## 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.

Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-8. What is a remote method?

The LoopBack standard model REST API includes the create, retrieve, update, and delete operations on model properties. For more information, see:  
<https://docs.strongloop.com/display/public/LB/PersistedModel+REST+API>

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

}
```

common/models/**product.js**

*Figure 5-9. 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 invokes 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 access 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

Figure 5-11. Example: Remote hook

This remote hook example is taken from exercise 3. In this scenario, the user creates a item review with the POST /review API operation. After the create operation completes, LoopBack executes 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.

Not shown on this slide is the code that updates the average rating value in the item model.

## 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 invokes 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.
- Operation hook is an advanced topic that requires an understanding of the model API and the underlying LoopBack classes that implement the API.
- For more information, see:  
<https://docs.strongloop.com/display/public/LB/Operation+hooks>

## 5.2. Node.js application deployment with Collectives

## Node.js application deployment with Collectives

Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-13. Node.js application deployment with Collectives

## Topics

- LoopBack remote methods and hooks
- ▶ Node.js application deployment with Collectives

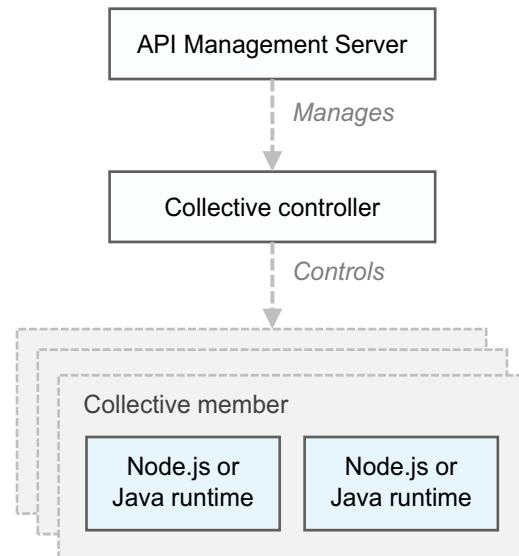
Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-14. Topics

## Manage API runtime environments with Collectives

- A collective is a group of multiple servers in a single management domain.
- Collectives provide:
  - Lightweight cluster support
  - A common point for automated deployment to multiple host systems
  - A secure, scalable, highly available operations point of control
  - A simple control point for monitoring operational status



*Figure 5-15. Manage API runtime environments with Collectives*

A collective is a group of multiple servers from a single management domain. The API Connect collective controller manages member servers in the collective. Each collective member hosts a server runtime environment for API implementations. The server runtime environment for API implementations is known as a compute runtime.

## Collectives

- A **collective** is an administrative domain for a collection of *collective members*.
- A **collective controller** is a control point for a collective.
  - The controller holds information about *collective members*.
  - A collective controller provides a REST interface.
  - You configure a collective with one or more controllers.
- A **collective member** is a server in a collective.
  - A collective member is responsible for a number of Node runtimes: one for each deployed application.
  - A collective member shares information about itself with *collective controllers* in the same *collective*.
  - This information includes network location, security information, and operational status.
- You use the **admin center** to connect to a collective controller and query information about collectives and manage members in a collective.

## Collectives and API Connect

- In an API Connect solution, you create collectives to deploy and run:
  - API implementations
  - Micro gateway
- After you developed and tested API implementations in your development environment, you deploy your application to collectives.
- You register the collective controller, collective member, and applications within each member with the API Connect Cloud Manager.

Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-17. Collectives and API Connect

The **API Connect collective controller** and the **API Connect collective member** are software components from the API Connect product. You download the installation media for the controller and member from the same place as API Connect software.

For the Bluemix service and Bluemix dedicated versions of API Connect, see the product documentation for details on where to download the collective controller and member software.

## How to deploy an API application to a collective

1. Install and set up a collective controller.
2. Install and set up a collective member.
3. Add the host to a collective with a collective controller.
4. Register the API application with the API Connect API Manager.
5. Publish the Node.js LoopBack API application to the API Manager.

## Step 1: Install and set up a collective controller

1. Download the API Connect collective controller from Passport Advantage website.
2. Install the collective controller.

```
$ npm install -g --unsafe-perm /<path>/apiconnect-collective-controller-<platform>-<version>.tgz
```

3. Configure and start the collective controller.

```
$ wlpn-controller setup --password=<controller-password>
Setting up the controller, this may take a few minutes.
Setting up collective...
Set up complete!
---
Default Liberty Admin User: root
This user account is required to log into the admin center
---
To start your controller, run `wlpn-controller start`.
$ wlpn-controller start
```

Figure 5-19. Step 1: Install and set up a collective controller

The *platform* and *version* values depend on the operating system of your host server, and the software level. The *path* represents the relative path to the API Connect Collective Controller archive that you downloaded.

## Step 2: Install and set up a collective member

1. Download the API Connect collective member host from Passport Advantage website.
2. Install the collective member.

```
$ npm install -g --unsafe-perm /<path>/apiconnect-collective-member-<platform>-<version>.tgz
```

Figure 5-20. Step 2: Install and set up a collective member

You can also install the API Connect collective member software through the public NPM repository. You can also point the Node package installer to an on-premises installation of API Connect Manager server for the installation package. See the knowledge center for more details: [https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.overview.doc/tapic\\_install\\_collective.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/tapic_install_collective.html)

## Step 3: Add the host to a collective

- If you set up a host on the same server as the collective controller, add the host to the collective with the updateHost command.

```
$ wlpn-controller updateHost myhost01 --host=localhost --
port=9443 --user=root --password=<controller-password> --
autoAcceptCertificates
Successfully registered/updated host: myhost01
```

- Alternatively, if you deploy the host on a different server than the collective controller, use the registerHost command.

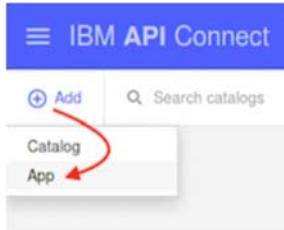
```
$ wlpn-controller registerHost myhost01 --host=localhost --
port=9443 --user=root --password=<controller-password> --
autoAcceptCertificates
Successfully registered/updated host: myhost01
```

Figure 5-21. Step 3: Add the host to a collective

As a recommended practice, set up a secure shell connection (SSH) on the API Connect collective controller server. You must provide additional parameters in the registerHost or updateHost commands for a secure connection. For example, you specify a path to the SSH private key that the controller uses to authenticate the host. See the knowledge center for more information: [https://www.ibm.com/support/knowledgecenter/SSMNED\\_5.0.0/com.ibm.apic.overview.doc/topic\\_install\\_collective.html](https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/topic_install_collective.html)

## Step 4: Register the application with API Manager server

1. Open a web browser to the API Connect **API Manager**.
2. Log in to API Manager with an
3. Select **Add App** wizard in API Manager.



4. After you create the provider application, copy the **App identifier** variable.

### App Identifier

This identifier can be used to configure the 'app' variable in the Developer Toolkit



Figure 5-22. Step 4: Register the application with API Manager server

The API Connect API Manager is one of six components in your API Connect solution. The API Manager keeps track of the components that you deploy in API Connect: the API Gateway, published APIs, and deployed API implementations. In the context of API Manager, an **app** is the implementation for an API. This concept is different from a third-party application, which consumes that APIs that you published.

In this step, log into the API Manager with a organization owner or administrator account. Define a provider app with the **Add App** wizard. After you create the app, examine the app details in the catalog. Copy the unique identifier for the app.

## Step 5: Publish the application to the API Manager

1. In the collective member host, navigate to the directory with the Node.js LoopBack API implementation.
2. Set the *apic* configuration to your provider app.

```
$ apic config:set app=<app-identifier>
```

3. Log in to the API Connect API Manager server.

```
$ apic login --type app
```

4. Publish the app to the API Manager server.

```
$ apic apps:publish
```

## Unit summary

- Explain the purpose of a remote method
- Explain the purpose of a remote and operation hook
- Explain the role of Collectives in Node.js application deployment
- Explain how to deploy LoopBack API applications with a Collective Controller

Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-24. 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. 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.



Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-25. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: The API Manager does not host the LoopBack application: the collective member host runs the application.  
The answer is True.
  
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.



## Exercise: Customize and deploy a LoopBack application

### Lab 3

Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-27. Exercise: Customize and deploy a LoopBack application

## Exercise objectives

- Define a remote hook in a LoopBack application
- Publish a LoopBack application to an API Connect Liberty Collective



Implement and publish LoopBack API applications

© Copyright IBM Corporation 2016

Figure 5-28. Exercise objectives

---

# Unit 6. Secure an API with security definitions

## Estimated time

01:30

## Overview

In this unit, you explore how to secure your API at the API Gateway with web application authentication and authorization standards. You learn how to configure API keys in API definitions. You also learn how to authenticate API clients with HTTP Basic authentication and the OAuth 2.0 authorization schemes.

## Unit objectives

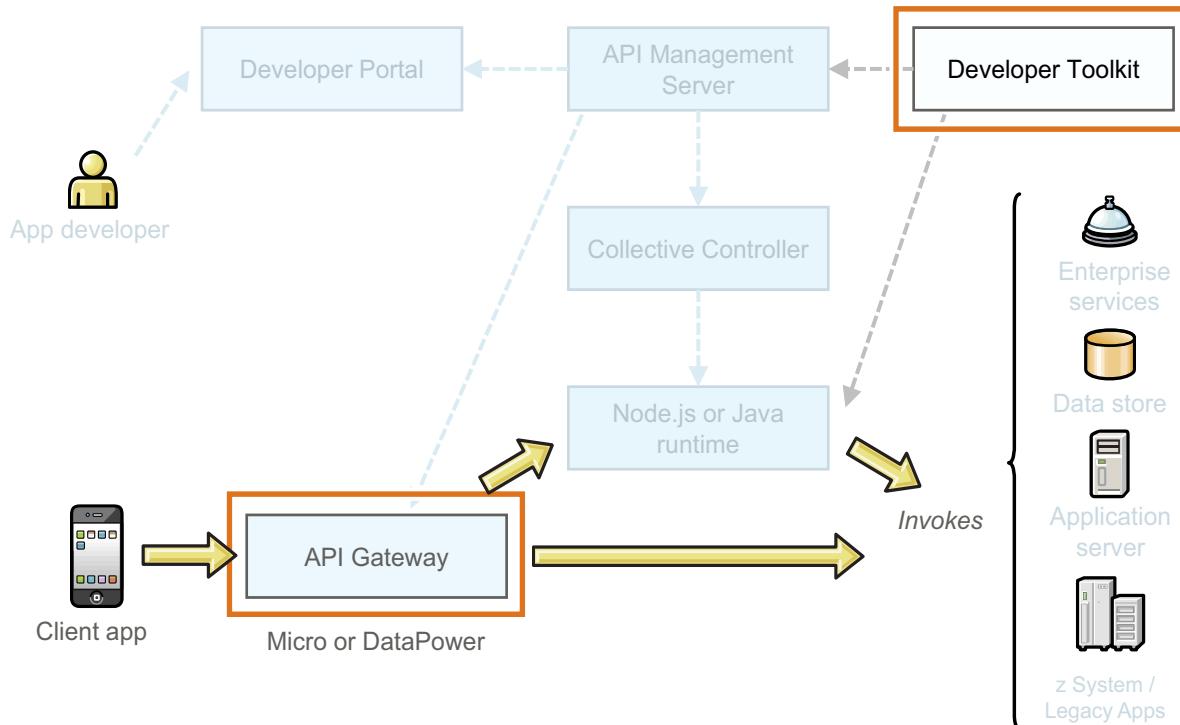
- Identify the security definition options in API Connect
- Explain the concept and use cases for API keys
- Explain the concept and use cases for HTTP Basic Authentication
- Explain the concept and use cases for OAuth 2.0
- Identify OAuth 2.0 roles, client types, and schemes
- Explain the difference between confidential and public schemes
- Identify the OAuth 2.0 provider API settings
- Identify the OAuth 2.0 security definition settings
- Describe how to acquire and use OAuth 2.0 access tokens

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

*Figure 6-1. Unit objectives*

## API Connect: Topology view



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-2. API Connect: Topology view

The topology view displays how the six components of IBM API Connect relate to one another.

This unit focuses on the LoopBack generation features in the `apic` command-line utility. The utility is part of the API Connect Developer Toolkit.

## 6.1. API security concepts

## API security concepts

Secure an API with security definitions

© Copyright IBM Corporation 2016

*Figure 6-3. API security concepts*

## Topics

### API security concepts

- Identify client applications with API key
- Authenticate clients with HTTP basic authentication
- Introduction to OAuth 2.0
- Configure OAuth 2.0 security in API Connect

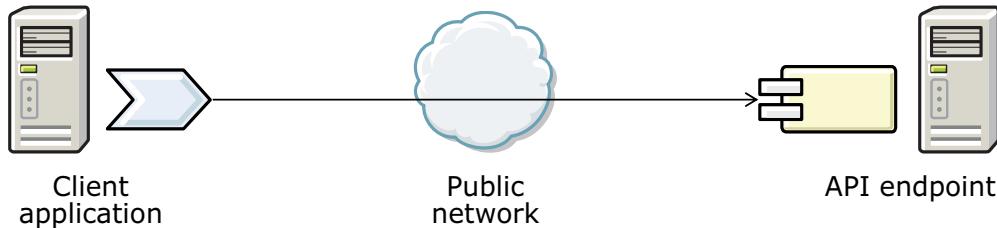
[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-4. 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



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-5. Authentication and authorization: API Security Definitions

This unit discusses how to authenticate and authorize API clients with IBM API Connect. There are other aspects of API security that you must consider, but those aspects are not covered by API security definitions. For example, encryption and integrity are not covered by API security definitions. You define transport level security (TLS) providers in the IBM API Management Server.

## 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.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

*Figure 6-6. 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.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-7. What types of security definitions can you define?

In API Connect, the three security definitions configure client authentication and authorization for API clients. Other security aspects, such as message encryption, are not covered in the security settings in your API definition.

## 6.2. Identify client applications with API key

## Identify client applications with API key

Secure an API with security definitions

© Copyright IBM Corporation 2016

*Figure 6-8. 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
- Configure OAuth 2.0 security in API Connect

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-9. Topics

## API key: a unique client application identifier

The screenshot shows the 'Security Definitions' section of the API Connect developer portal. On the left, there's a sidebar with various navigation items like Info, Schemes, Host, etc., and a 'Security Definitions' item which is highlighted with an orange border. The main area displays a table for a security definition named 'clientID (API Key)'. The table has rows for Name (clientID), Parameter name (X-IBM-Client-Id), Located In (Header), and Description. The 'clientID (API Key)' row is also highlighted with an orange border.

Security Definitions	
<b>clientID (API Key)</b>	
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**.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-10. 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.

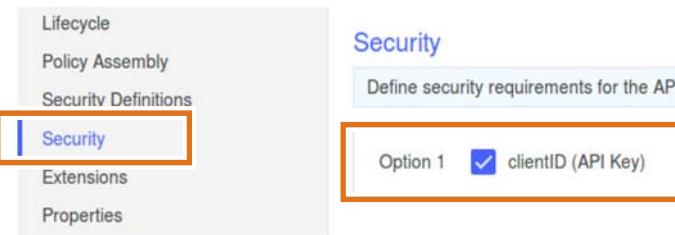
The client secret is a unique value that API Connect generates. When a client developer registers an application, the API Connect developer portal creates and provides the client ID and client secret.

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 client secret. When you create an API definition, the API Designer adds a security definition for a **client ID** for you.

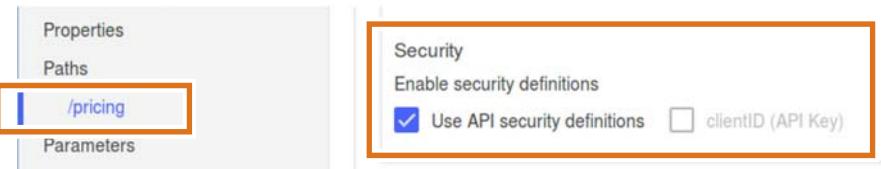


## Enforcing security definitions

- 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 applied a security definition, you can enable or clear the security requirement on an API operation level in the **paths** API definition section.



[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-11. 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 6-13. Example: client ID and client secret in the message header*

The names of the HTTP headers and the query parameters are the default names set by API Connect. 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.

## 6.3. Authenticate clients with HTTP basic authentication

## Authenticate clients with HTTP basic authentication

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-14. 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
  - Configure OAuth 2.0 security in API Connect

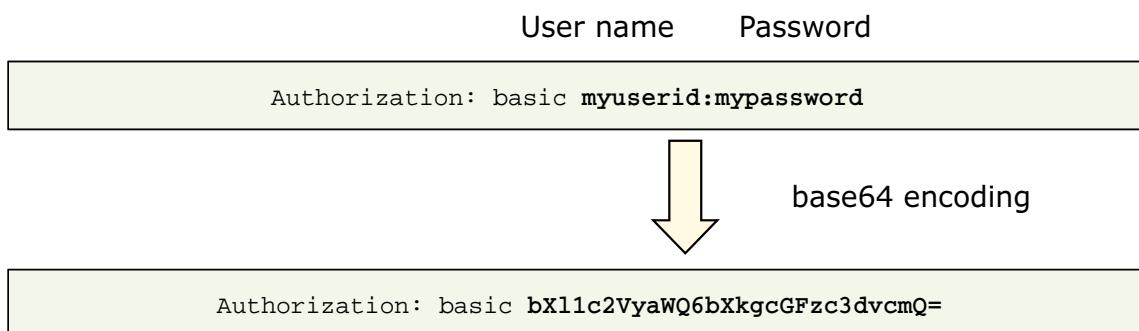
[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-15. 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 does not protect the contents of the message.



[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-16. 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 by a colon. Before the client sends the request message, it encodes the user name, colon, and password with the base64 encoding scheme.

Keep in mind that this encoding scheme is not encryption – anyone who intercepts the message can decode the message and retrieve the user name and password.

Base64 is a scheme for encoding binary data as text. The most common use of Base64 is to encode photo, video, and document attachments to email.

## Example: storing credentials in HTTP request header

```

PUT /api/employee HTTP/1.1
Host: www.example.com/hr/
Authorization: basic bX11c2VyaWQ6bdvcmQ=
Date: Mon, 12 Dec 2016, 15:35:12 GMT

{
  "fname" : "John",
  "lname" : "Smith",
  "email" : "jsmith@example.com",
  "dept" : "finance",
  "country" : "Canada"
}

```

The diagram illustrates the structure of an HTTP request message. It shows a left-hand side with the raw request text and a right-hand side with descriptive labels and brackets:

- HTTP basic authorization header:** Brackets group the `Authorization: basic bX11c2VyaWQ6bdvcmQ=` line.
- JSON data that is submitted in a REST service request:** Brackets group the JSON object containing employee information.
- HTTP header:** Brackets group the first two lines of the request (method, host).
- HTTP body:** Brackets group the entire bottom section of the request, from the date to the closing brace.

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-17. Example: storing credentials in HTTP request header

The HTTP basic authentication header appears in the top 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

Policy Assembly

**Security Definitions** ← 1

Security

Extensions

Properties

Paths

- /products
- /products/{id}/exists
- /products/{id}
- /products/findOne
- /products/update
- /products/count
- /products/change-stream
- /warehouses

exampleBasicAuth (Basic)

Name ← 2

An example of a Basic Authentication security definition.

Authenticate using

Authentication URL ← 3

URL ← 4

https://example.org/authentication

TLS Profile ← 5

example-tls-profile

User Registry ← 6

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-18. 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 an 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. An authentication service returns an HTTP status code of 200 success if the client sent a valid user name and password. Otherwise, the service returns an 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.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

*Figure 6-19. Message confidentiality with Transport Layer Security*

Although the HTTP basic authentication protocol does not require an encrypted connection, it is highly recommended 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. It merely assumes that you are who you say you are if you provide the correct user name and password. 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. It is highly recommended that your website use TLS 1.0 as the minimum version for securing HTTP connections.

## 6.4. Introduction to OAuth 2.0

## Introduction to OAuth 2.0

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-20. 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
- Configure OAuth 2.0 security in API Connect

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-21. 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 the user 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.
- It allows a user to grant access to different resources to different third-party application, for a specified duration of time.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

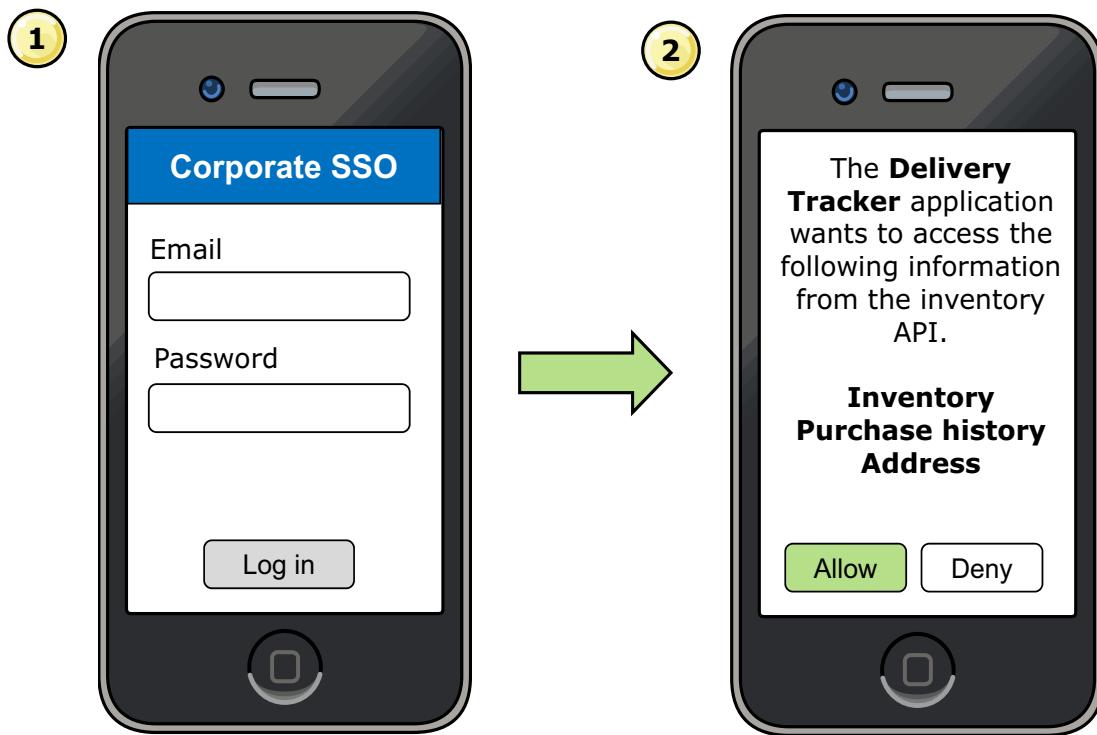
Figure 6-22. What is OAuth?

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 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 perform 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.

## Example: allow third-party access to shared resources



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-23. Example: allow third-party access to shared resources

Whenever you sign up for a web-based application or a mobile application, you create a new account on the server with a user name and password. The process becomes tedious for the user 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.

There are four participants in this scenario. 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, the authorization service grants an access token to the application. At no time does the third-party application see your user name or password on the authorization service.

## Example: third-party access to inventory API resources

- OAuth allows social network applications to share resources.



Alice is the **owner** of inventory records. As a **resource owner**, Alice declares which applications can access the inventory API on her behalf.



The inventory API provides online access to inventory records. This service acts as the **resource server**. It manages access to Alice's records.



Delivery tracker, a third-party **client application**, wants to access Alice's inventory records from the API.



An **authorization 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.

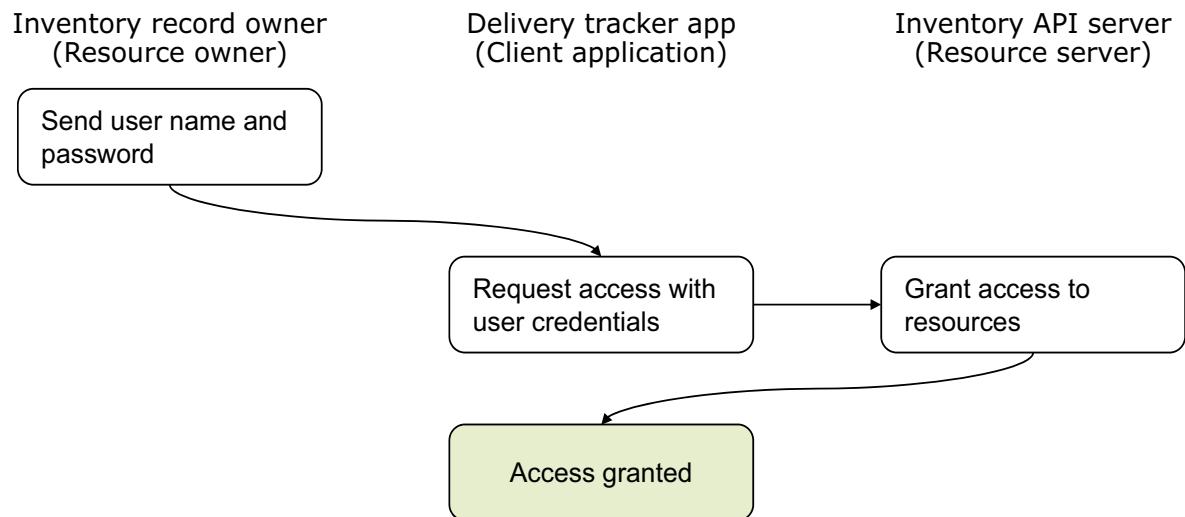
[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-24. 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 an 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.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

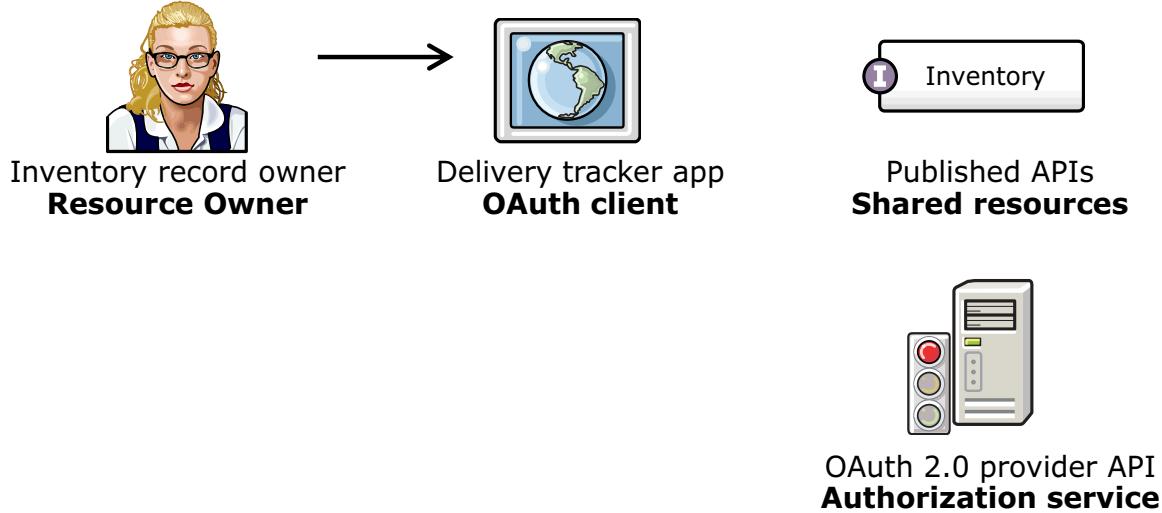
Figure 6-25. Before OAuth – sharing user passwords

Without OAuth, the user must give their user name and password to the third-party application. In turn, the third-party application sends these credentials while posing as the user. For convenience's sake, the application saves a copy of the user name and password.

There are several issues with this scenario. 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, there is no simple way to revoke access for one particular third-party application. The user must reset their password, which breaks access from all third-party applications.

## OAuth Step 1: Resource owner requests access

- Step 1: Alice, as the resource owner, requests access to the online photo album to the OAuth client.



[Secure an API with security definitions](#)

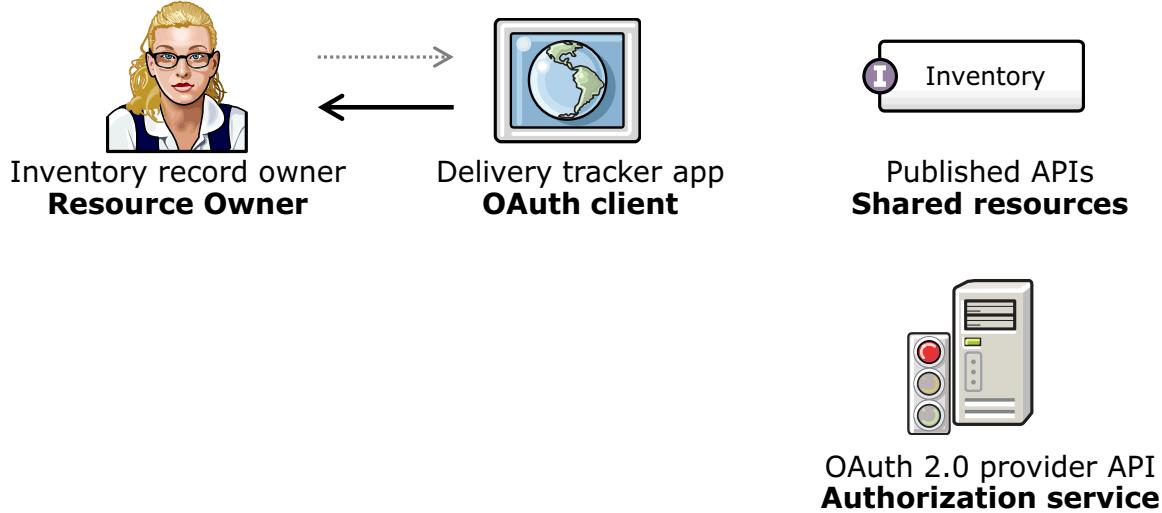
© Copyright IBM Corporation 2016

Figure 6-26. OAuth Step 1: Resource owner requests access

In this scenario, Alice is the owner of a photo album that is hosted on an online photo service. Alice wants to print a set of photos with a photo printing service. Alice is the resource owner, and the photo printing service is a third-party OAuth client application. Alice starts the process when she selects the "print from my photo album" 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.



[Secure an API with security definitions](#)

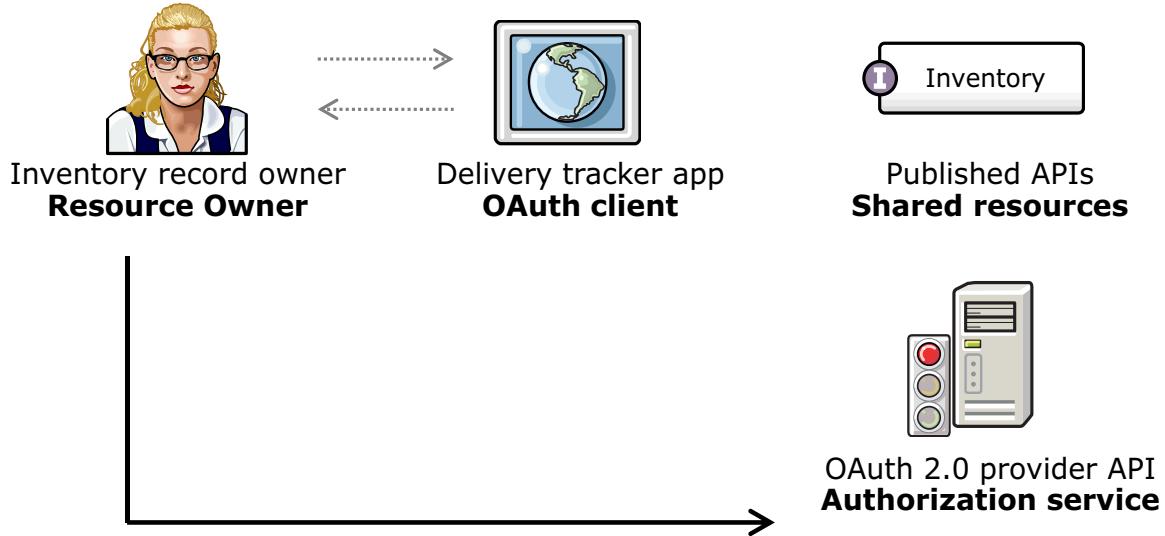
© Copyright IBM Corporation 2016

Figure 6-27. 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 her online photo album. 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.



[Secure an API with security definitions](#)

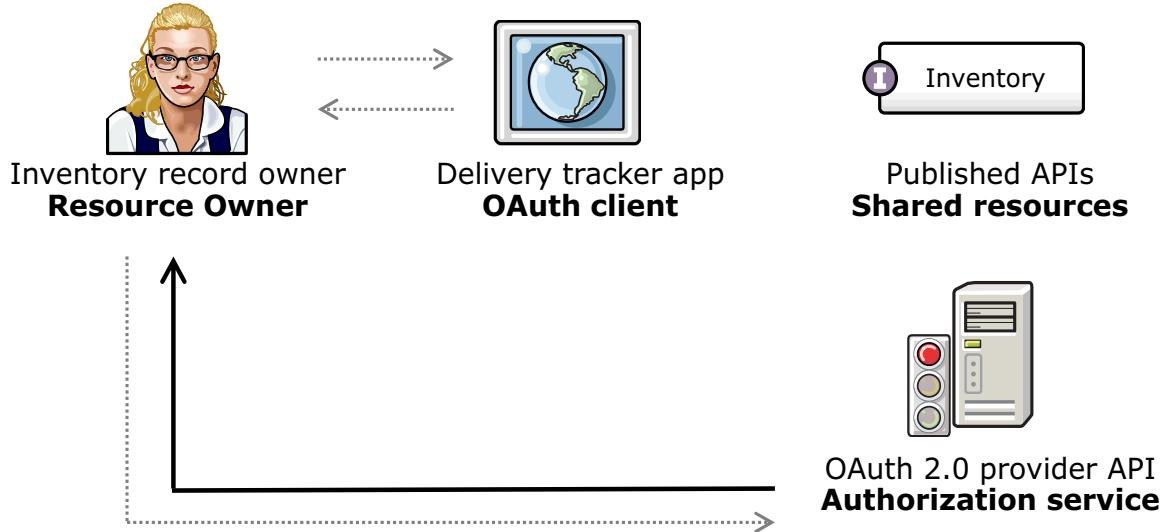
© Copyright IBM Corporation 2016

Figure 6-28. 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.



Secure an API with security definitions

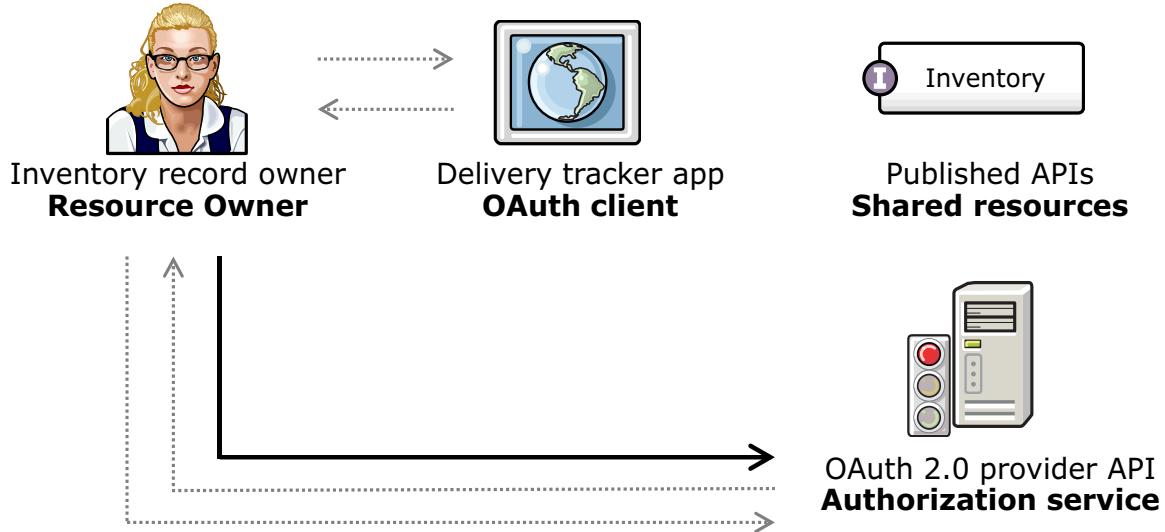
© Copyright IBM Corporation 2016

Figure 6-29. 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.

## OAuth Step 5: Resource owner grants client access to resources

- Step 5: The resource owner submits the form to allow or to deny access.



Secure an API with security definitions

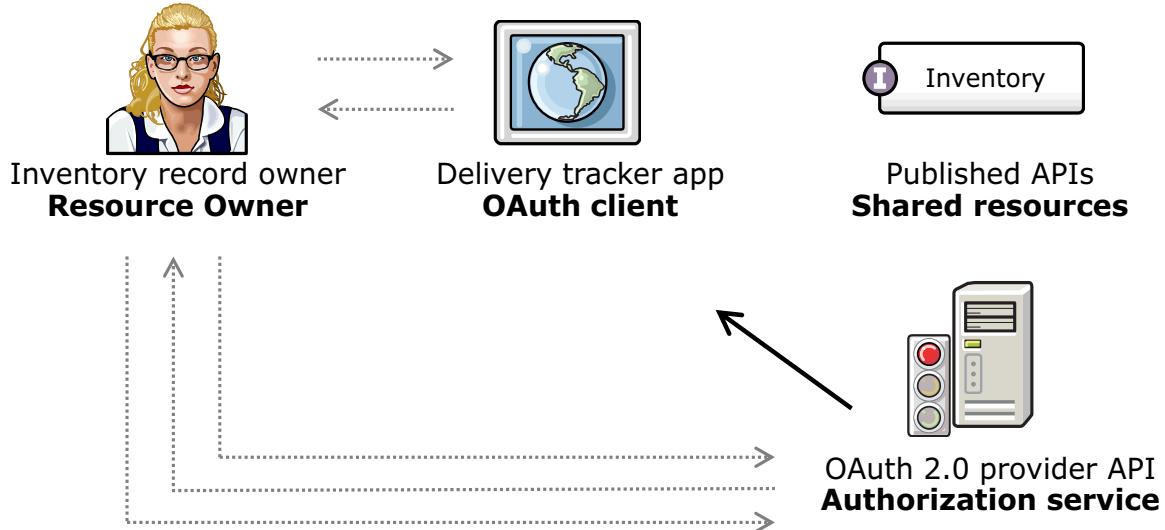
© Copyright IBM Corporation 2016

Figure 6-30. OAuth Step 5: Resource owner grants client access to resources

The resource owner, Alice, submits the web form to allow or deny access to her resources.

## OAuth Step 6: Authorization 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.



Secure an API with security definitions

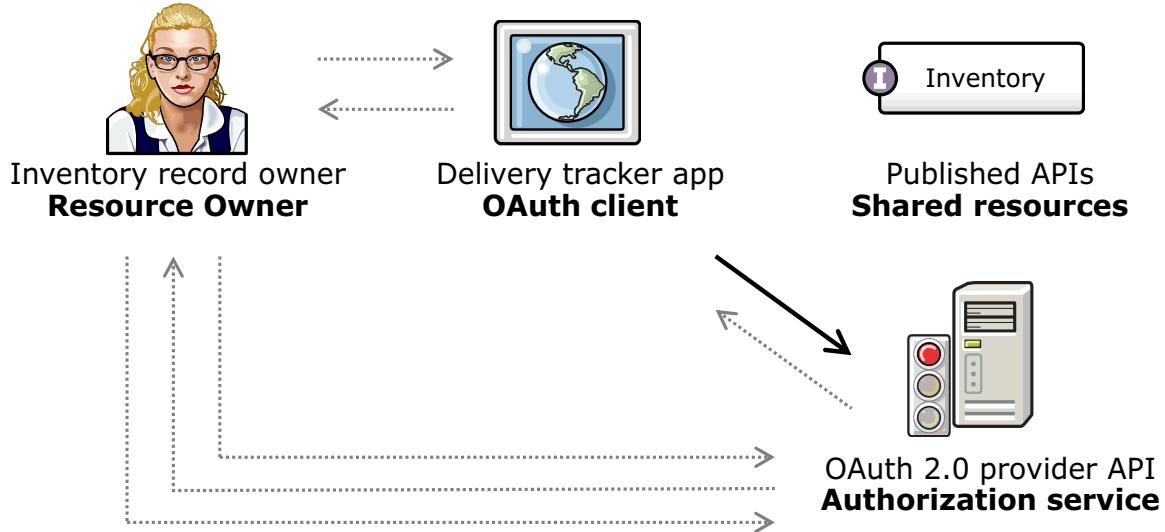
© Copyright IBM Corporation 2016

Figure 6-31. 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 allows the OAuth client 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.



Secure an API with security definitions

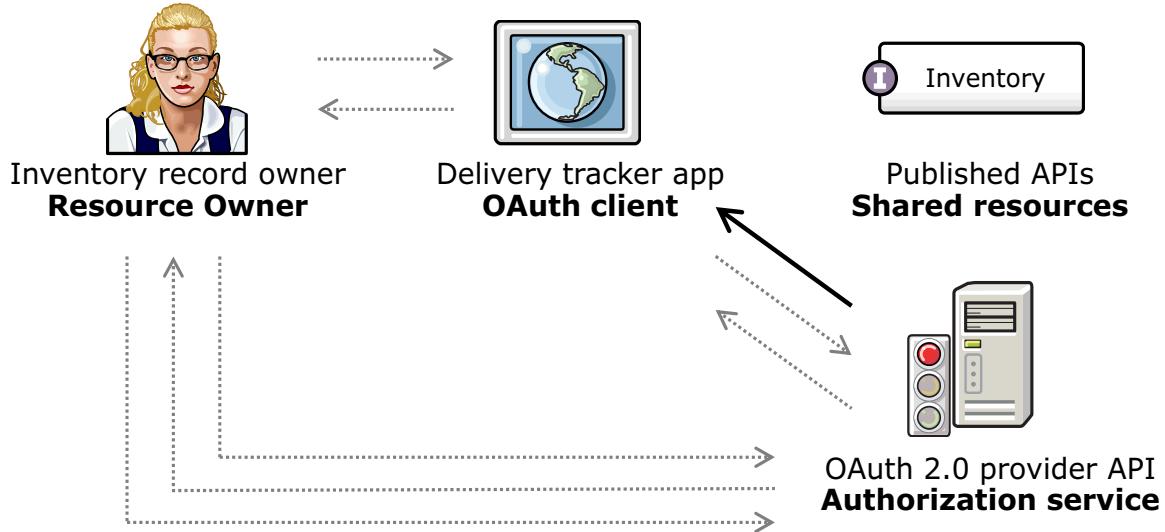
© Copyright IBM Corporation 2016

Figure 6-32. 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.



Secure an API with security definitions

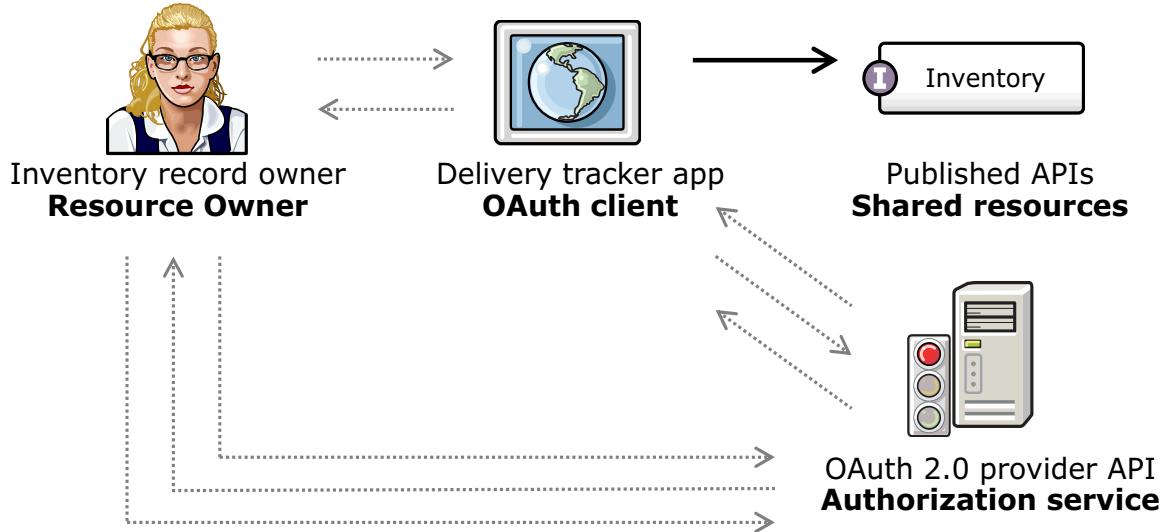
© Copyright IBM Corporation 2016

Figure 6-33. 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.



Secure an API with security definitions

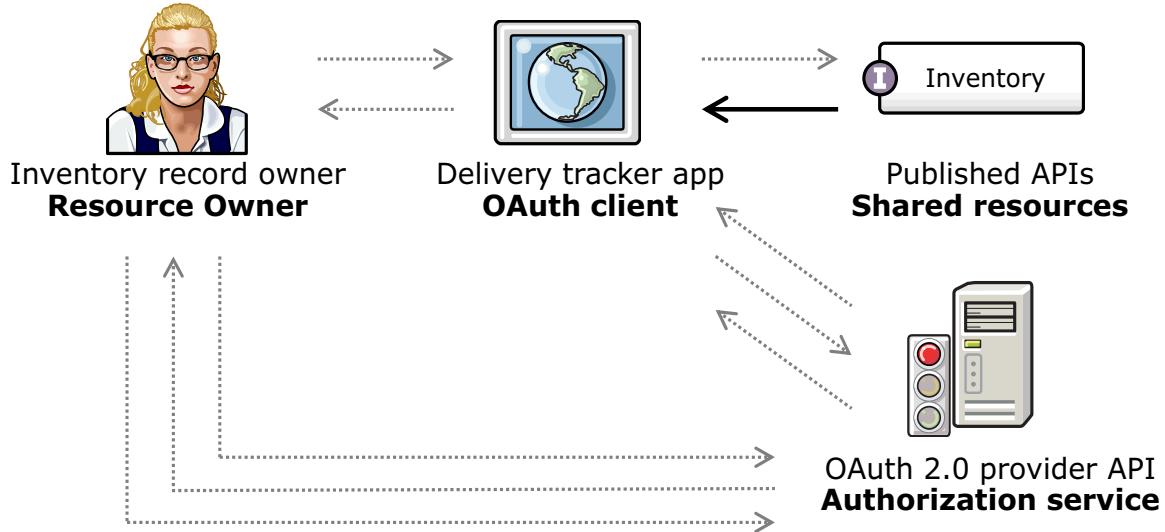
© Copyright IBM Corporation 2016

Figure 6-34. 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.



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-35. 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.

## 6.5. Configure OAuth 2.0 security in API Connect

## Configure OAuth 2.0 security in API Connect

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-36. Configure OAuth 2.0 security in API Connect

## Topics

- API security concepts
  - Identify client applications with API key
  - Authenticate clients with HTTP basic authentication
  - Introduction to OAuth 2.0
-  Configure OAuth 2.0 security in API Connect

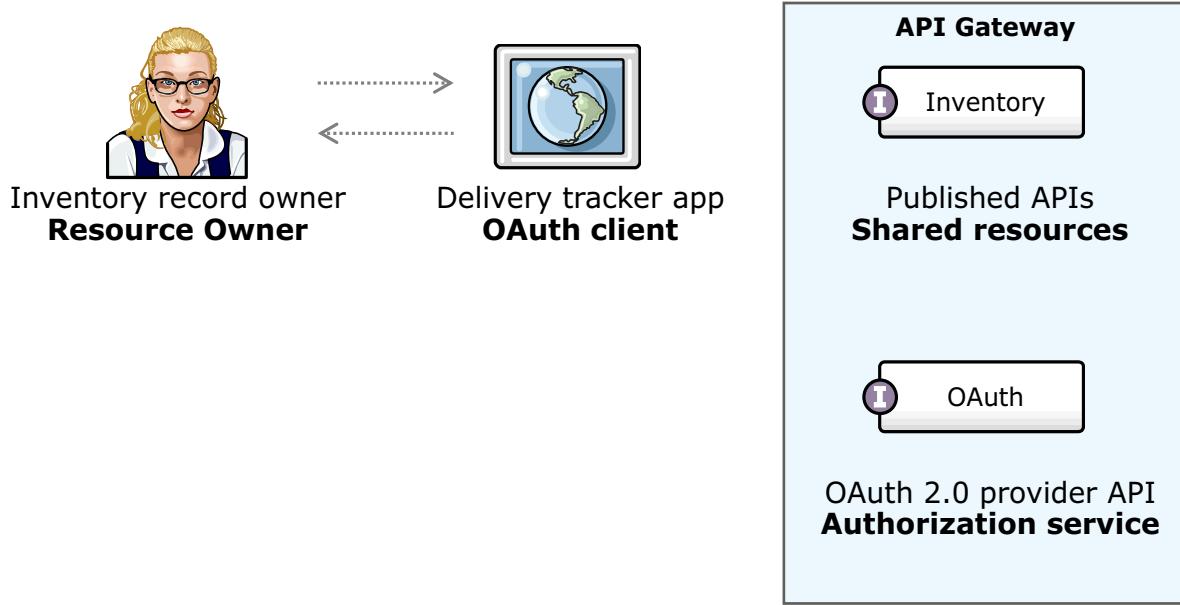
[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-37. Topics

## 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**.



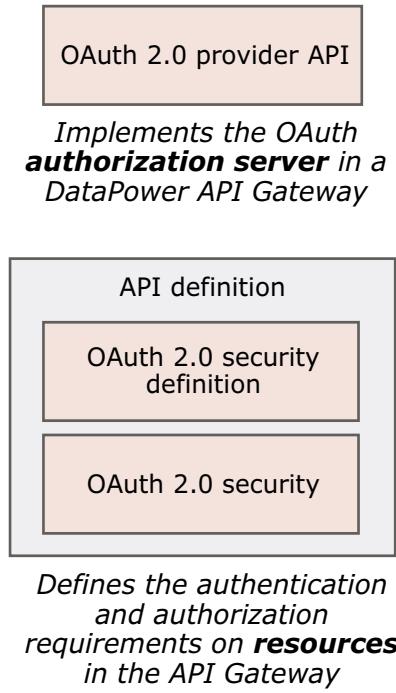
Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-38. 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.

## How to enable OAuth 2.0 authentication



- 1. Create an OAuth 2.0 Provider API.**
  - The OAuth 2.0 provider serves authorization services to one or more APIs that employ OAuth security definitions.
- 2. Configure the OAuth 2.0 Provider API settings.**
- 3. For each API, create an OAuth 2.0 security definition.**
  - The OAuth 2.0 security definition states the type of **OAuth flow** and **scope**.
- 4. In the security section, select the OAuth 2.0 security definition.**
  - Enable the OAuth 2.0 security definition on one or more operations in the API.

[Secure an API with security definitions](#)

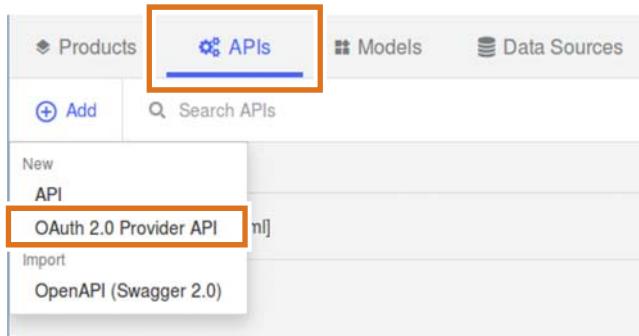
© Copyright IBM Corporation 2016

Figure 6-39. How to enable OAuth 2.0 authentication



## Step 1: Create an OAuth 2.0 provider

1. In API Designer, select the **API** tab to list all APIs.
2. Select **New > 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 an API product.

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-40. Step 1: Create an OAuth 2.0 provider



## Step 2: Configure the OAuth 2.0 provider settings

### 5. Review the OAuth 2.0 settings in the OAuth 2.0 provider API.

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-41. 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 only view bank accounts that they own. 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 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 guarantee 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.

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-42. 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 guarantee 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>

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

Figure 6-43. OAuth 2.0 Provider API: OAuth flow and grant types

There are four options for OAuth 2.0 authorization grant types.

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 to resources that are previously arranged with the resource server. This grant type is available to confidential client types only.



## Step 2: Authorization and token settings

**Identity extraction**

Collect credentials using  
Default form

**Authentication**

Authenticate application users using  
Authentication URL: <https://example.com/auth/url>

**Authorization**

Authorize application users using  
Authenticated

**Tokens**

Access tokens

Time to live (seconds): 3600

Enable refresh tokens

Count: 2048

Time to live (seconds): 2682000

Enable revocation URL

Revocation URL: <http://example.com/access/revocation/>

TLS Profile: example-tls-profile

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-44. Step 2: Authorization and token settings

Scroll down to reveal the rest of the OAuth 2.0 provider API settings.

5. In order 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 log in web site.
6. The **authentication** settings 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 allow access.
8. The **access token** settings determine how long to keep tokens alive, and whether to enable a renewal service for access tokens.
9. The **revocation URL** setting allows the resource owner 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.

## Step 3: Create an OAuth 2.0 security definition

- The **OAuth 2.0 provider API**: the **authorization server** for your OAuth authentication flow.
- 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, select the **APIs** tab.
2. Select an existing API definition or create a new API definition.

The screenshot shows the IBM API Designer interface. At the top, there are tabs: 'Products', 'APIs' (which is highlighted with a blue border), 'Models', and 'Data Sources'. Below the tabs, there are buttons for '+ Add' and 'Search APIs'. The main area displays a list of API definitions. The first item is 'example-oauth-provider 1.0.0 [example-oauth-provider\_1.0.0.yaml]'. The second item, 'pricing 1.0.0 [pricing.yaml]', is highlighted with an orange border. The background of the interface is light gray, and the overall design is clean and modern.

Secure an API with security definitions

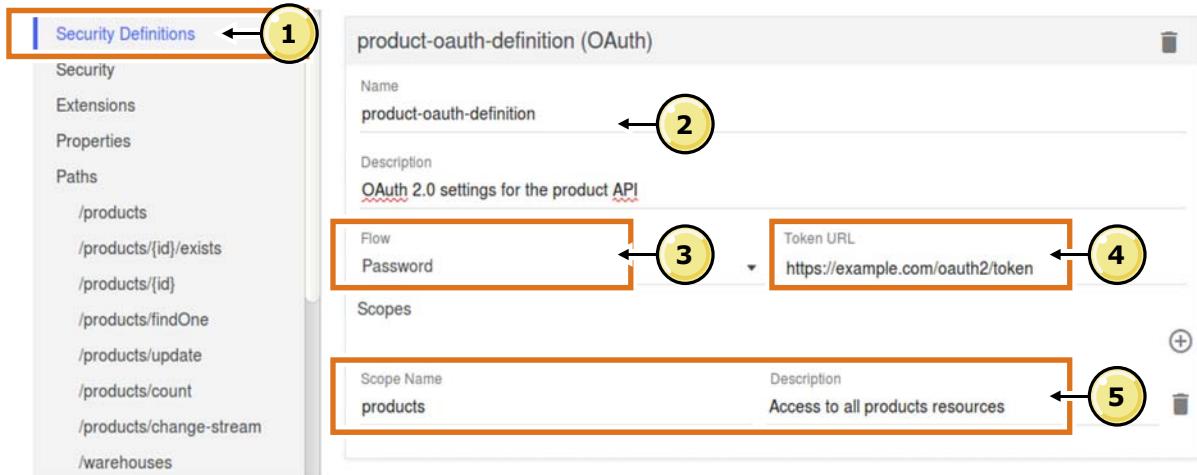
© Copyright IBM Corporation 2016

Figure 6-46. Step 3: Create an OAuth 2.0 security definition

The recommended practice is to keep the OAuth 2.0 provider API separate from your business APIs.

## Step 3: 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



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-47. Step 3: 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

- 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
- Identify OAuth 2.0 roles, client types, and schemes
- Explain the difference between confidential and public schemes
- Identify the OAuth 2.0 provider API settings
- Identify the OAuth 2.0 security definition settings
- Describe how to acquire and use OAuth 2.0 access tokens

[Secure an API with security definitions](#)

© Copyright IBM Corporation 2016

*Figure 6-48. 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. 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 guarantee that they can keep a client password secret.
  - D. Public clients do not guarantee that they can keep a client password secret.



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-49. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## 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.
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 guarantee that they can keep a client password secret.
  - D. Public clients do not guarantee that they can keep a client password secret.

The answer is B.

The answer is D.



## Exercise: Configure and secure an API

### Lab 4

Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-51. Exercise: Configure and secure an API

## Exercise objectives

- Create an OAuth 2.0 provider with the resource owner password grant type.
- Secure an existing API with an OAuth 2.0 provider
- Add catalog-specific properties to an API
- Assemble an API implementation with the activity-log, set-variable, and invoke policies



Secure an API with security definitions

© Copyright IBM Corporation 2016

Figure 6-52. Exercise objectives

---

# Unit 7. Assemble an API in the API Designer

## Estimated time

01:00

## Overview

In this unit, you assemble an API with the API designer. The API Gateway enforces non-functional requirements from an API, such as authentication, authorization, logging, auditing, message routing, and transformation. You define these message control and processing policies as assemblies in API Designer. You learn about the policies in the two types of API gateway: the micro gateway and the DataPower gateway. You learn how to build an assembly of message processing policies in the API Designer.

## How you will check your progress

- Lab exercise
- Review questions

## Unit objectives

- Describe the concept of API policies
- Identify the types of policies
- Describe the capabilities of the Micro Gateway policies
- Describe the capabilities of the DataPower gateway policies
- Describe how to add components in an assembly
- Describe the activity log policy
- Describe the invoke policy
- Describe the gateway script policy
- Describe the map policy
- Create an assembly for an API operation that calls an existing service

## 7.1. API policies

## API policies

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-2. API policies

## Topics

### API policies

- Assemble an API operation for an existing service

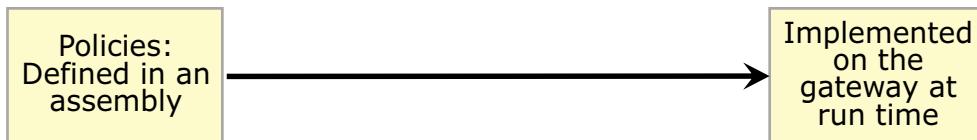
Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-3. Topics

## API policies

- API policies are configuration rules that determine how the API gateway processes API operation calls at run time
  - API policies and logic constructs are the building blocks of assembly flows
- API policies provide the means to configure capability
  - Logging, security, routing of requests, and data transformation
- Logic constructs affect which parts of the assembly are implemented
  - Grouped with policies in the assembly editor
  - Add logic constructs to change the flow through the assembly based on message contents and variables



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-4. API policies

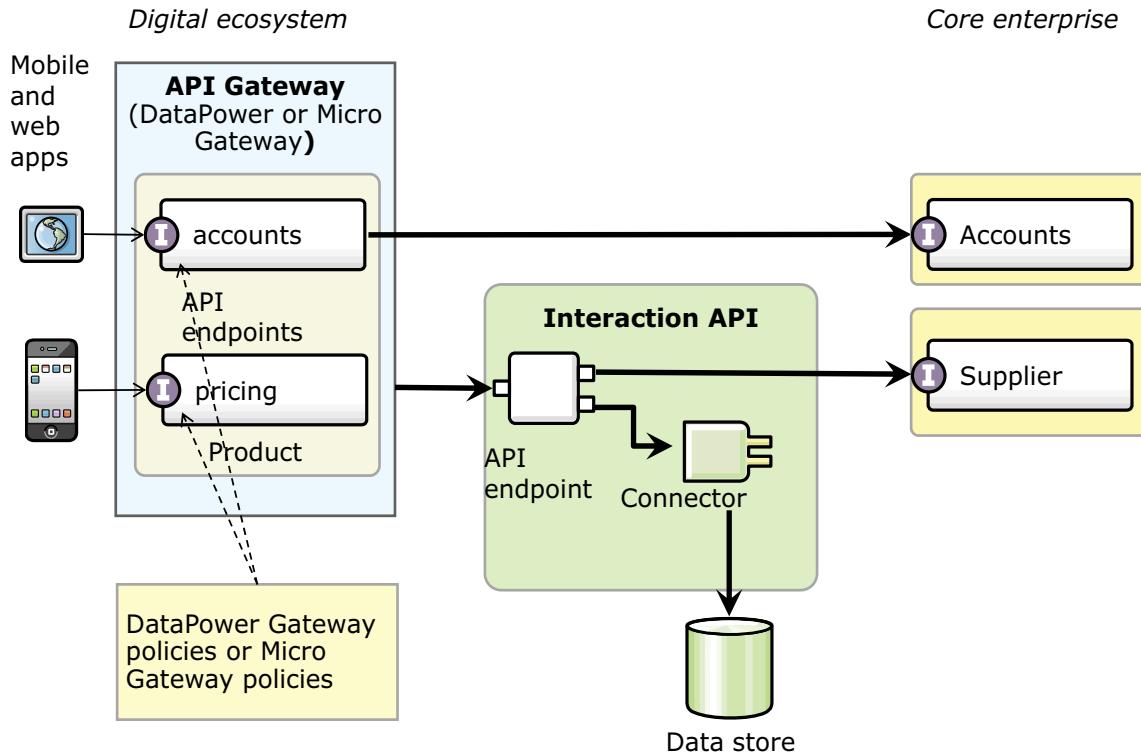
API policies and logic constructs are configuration rules that control a specific aspect of processing in the gateway server during the handling of an API invocation at run time.

Policies and logic constructs are the building blocks of assembly flows.

An assembly provides an implementation of the API operations of an API.

The policies and logic constructs that are defined in the assembly tell the runtime how to handle the request and response messages and how to handle any errors that occur.

## Message flows and policies in IBM API Connect



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-5. Message flows and policies in IBM API Connect

Policies are applied dynamically at runtime by the gateway server when the API endpoint is called.

DataPower Gateway policies are applied by the DataPower gateway.

Micro Gateway policies are applied by the Micro Gateway server.

These gateways enforce the rules that you apply in the policy.

## API policy types

- Built-in policies
  - Pre-configured policies in API Connect
  - Available from the palette of the Assembly editor in API Designer
- User-defined policies
  - Defined externally from API Connect
  - Imported into an API Connect catalog
  - Policy becomes available to be placed into an assembly in the Assembly editor

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-6. API policy types

IBM API Connect includes a number of built-in policies that you can use to apply preconfigured policy statements to an operation to control an aspect of processing in the gateway server when an API is invoked.

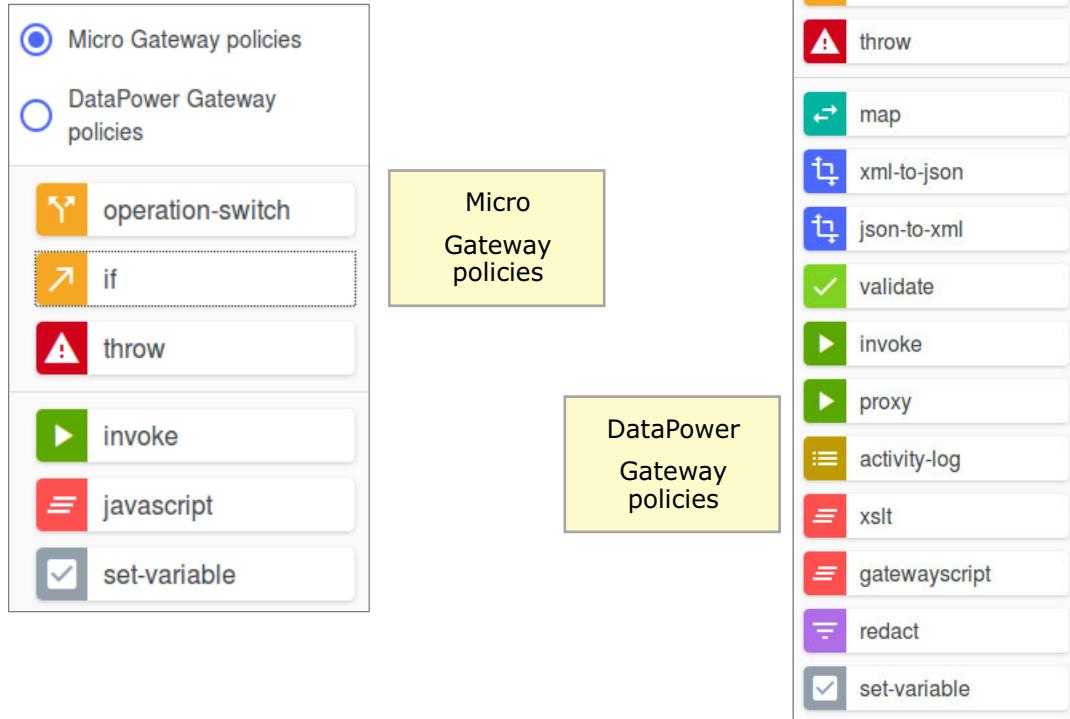
Built-in policies can be added from the palette of the assembly editor and configured in a message processing flow.

A user-defined policy is implemented in one of the following ways depending on the runtime platform:

- For an IBM DataPower Gateway, as a DataPower processing rule.
- For a Micro Gateway, as a Node.js module.



## Assembly logic and built-in policies



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-7. Assembly logic and built-in policies

Although some built-in policies can be used with both the DataPower Gateway and the Micro Gateway, some policies are restricted to a particular gateway.

The page shows the selectable assembly gateway policies for the Micro Gateway and the DataPower gateway.

A gateway is the runtime environment that an API endpoint is deployed to.

The gateway is responsible for running the API when the API operation is called.

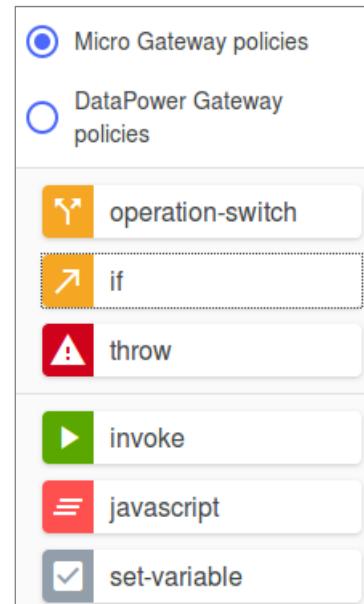
A Micro Gateway runs on a Node.js runtime, while the DataPower Gateway runs on the DataPower appliance runtime.

The Micro Gateway supports a subset of the DataPower Gateway policies.

These built-in policies enables you to apply a pre-configured policies to an assembly to control processing capabilities in the Gateway server. Built-in policies are applied by using the API Designer assembly editor to add the policy to your assembly and to configure the properties for that policy.

## Capabilities of gateway policies

- Focus is on building policies for IBM Connect Essentials edition that run on a Node.js runtime
  - Logic blocks
  - Set variables
  - JavaScript for Node.js
  - Invoke
- Built-in policies supported by Micro Gateway:
  - Client ID/Secret
  - Basic Authentication
  - Basic rate limiting
  - CORS
  - Invoke (call service over HTTP)
  - Set Variable
  - JavaScript invoke



[Assemble an API in the API Designer](#)

© Copyright IBM Corporation 2016

Figure 7-8. Capabilities of gateway policies

The Micro Gateway is a gateway that is built on Node.js, and provides enforcement for the authentication, authorization, and flow requirements of an API. The Micro Gateway is deployed on a API Connect Collective and has a limited number of policies available to it.

Built-in policies supported by Micro Gateway:

- Client ID/Secret
- Basic Authentication
- Basic rate limiting
- CORS (cross origin resource sharing)
- Invoke (call service over HTTP)
- Set variable
- JavaScript call

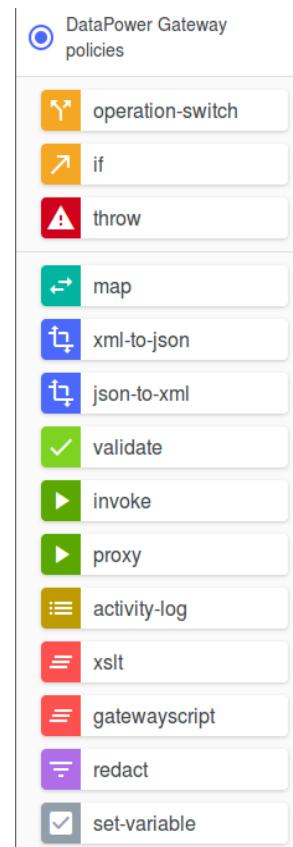
The Client ID/Client Secret, basic authentication, basic rate limiting, and CORS capabilities are not set within an assembly, but are configured within the API definition.

The invoke, set variable, and JavaScript functions are policies that can be used inside an assembly that is deployed to a Micro Gateway runtime.



## Capabilities of DataPower policies

- Focus is on building assemblies that control the message flow and content of APIs at run time
- Additional built-in policies supported by DataPower:
  - Oauth, advanced rate limiting, response caching (configured in the API definition, not assembly)
  - Mapping
  - Transformations (xml-to-json, json-to-xml)
  - REST validation (JSON)
  - Invocation (HTTP and HTTPS)
  - Proxy
  - Activity logging
  - XSLT invoke
  - Gateway script invoke
  - Redaction



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-9. Capabilities of DataPower policies

DataPower policies are the building blocks for assemblies that run on DataPower.

DataPower Gateway appliances help quickly secure, integrate, control and optimize access to a range of workloads through a single, extensible, DMZ-ready gateway.

These appliances act as security and integration gateways for a full range of mobile, cloud, web, and application programming interface (API) workloads.

Additional built-in policies supported by DataPower:

OAuth, advanced rate limiting, redaction, map, activity log, REST validation, proxy, gatewayscript invoke, XSLT invoke, JSON to/from XML transform, response caching.

Oauth, advanced rate limiting, and response caching are configured when the API is defined, and not set within an assembly.



## Adding components to your assembly (1 of 5)

1. From the API Designer, select the API that you want to work with

A screenshot of the API Designer interface. At the top, there are three tabs: "Products", "APIs" (which is highlighted with a blue underline), and "Models". Below the tabs is a toolbar with a "Add" button (containing a plus sign) and a "Search APIs" input field. The main area is titled "Title" and contains a list item: "hello-world 1.0.0 [hello-world.yaml]". This list item is highlighted with an orange rectangular border.

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-10. Adding components to your assembly (1 of 5)

From the API Designer select the API that you want to work with.



## Adding components to your assembly (2 of 5)

### 2. Click the **Assemble** tab

A screenshot of the API Designer interface. At the top, there are four tabs: "All APIs", "Design" (which is highlighted in blue), "Source", and "Assemble". The "Assemble" tab has a red box drawn around it. On the left, there is a sidebar with links: "Info", "Host", "Base Path", "Schemes", "Consumes", "Produces", and "..." (with an ellipsis). The main panel shows an "Info" section with the following details:

Title	hello-world
Name	hello-world
Version	1.0.0

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

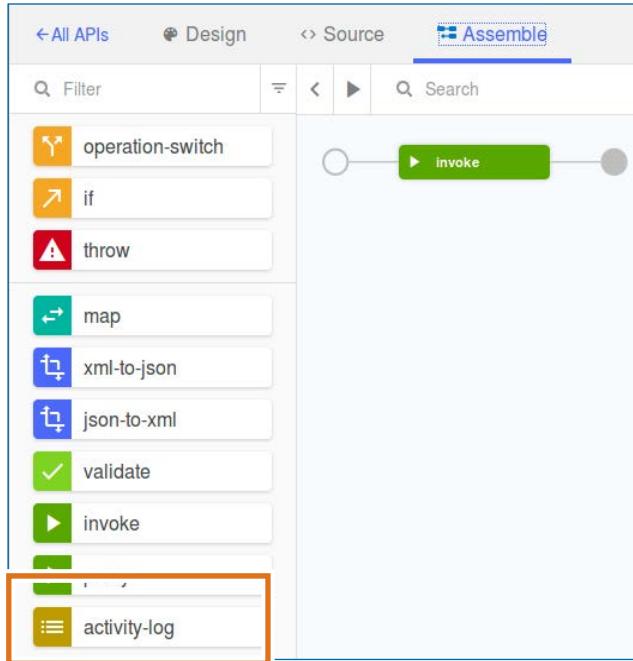
Figure 7-11. Adding components to your assembly (2 of 5)

Click the **Assemble** tab.

An assembly is formed of components that are applied to calls to and responses from operations in your API. Components can be either policies or logic constructs.

## Adding components to your assembly (3 of 5)

3. Find the policy that you want to add to the assembly



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-12. Adding components to your assembly (3 of 5)

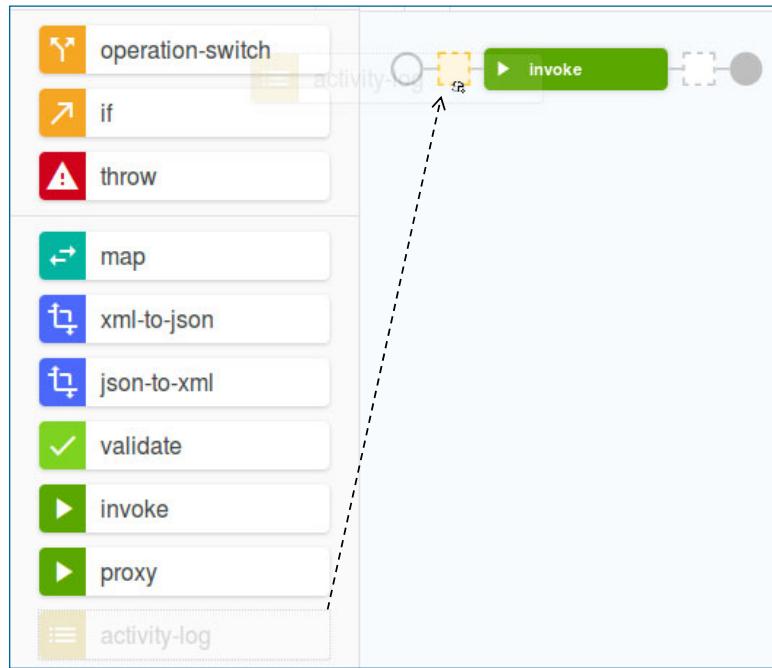
A default assembly with an Invoke policy is created in the flow area of the assembly.

Select the policy that you want to add from the Palette.

You can filter the policies in the palette according to the gateway or you can search for a policy.

## Adding components to your assembly (4 of 5)

4. Drop the policy that you want to add it to the assembly



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

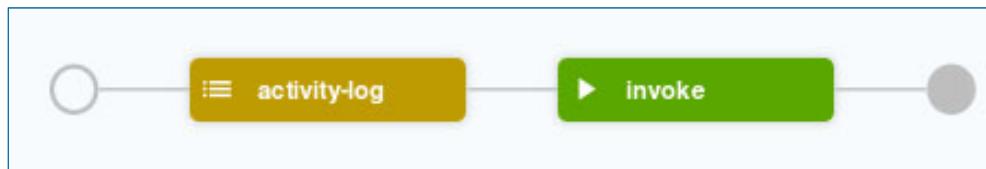
Figure 7-13. Adding components to your assembly (4 of 5)

Drag the component onto the canvas.

Drop the component in a dashed box to insert it into that position in the assembly.

## Adding components to your assembly (5 of 5)

4. The policy is added to the assembly



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

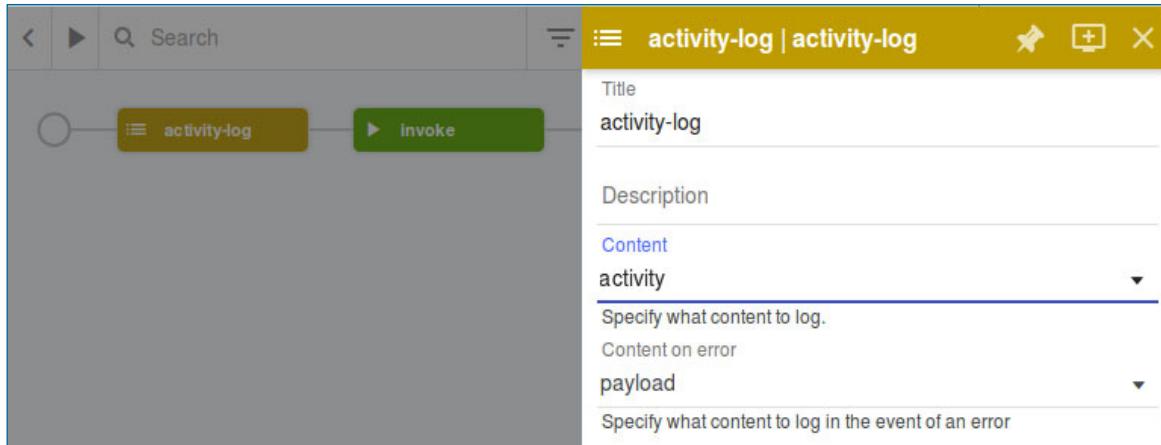
Figure 7-14. Adding components to your assembly (5 of 5)

The activity-log policy is added to the assembly before the operation is invoked.

This means that the request message can be logged according to the properties set in the activity-log properties sheet.

## Edit components in your assembly

- Click the component in the flow diagram
- Edit the properties in the property sheet



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

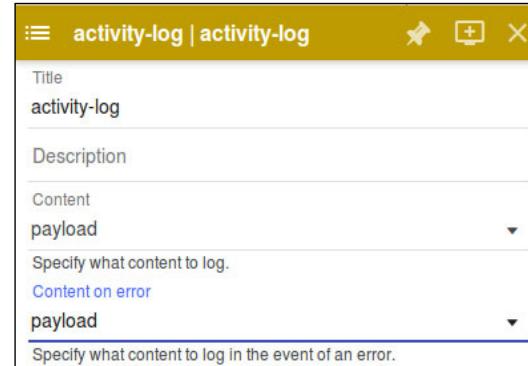
Figure 7-15. Edit components in your assembly

To edit the properties of the component:

- Click the component in the flow diagram. For example activity-log.
- The property sheet opens.
- Edit the properties in the property sheet.

## Activity log policy

- Use the activity-log policy to configure your logging preferences for the API activity that is stored in Analytics
- Policy properties are defined in the policy definition property sheet
- Policy values specified override the default settings for collecting and storing details of the API activity
- Properties
  - Title, description, content, content on error
- Values for content and content on error:
  - none – no logging occurs
  - activity – Logs invocation URL (only the resource URI is recorded)
  - header – Logs activity and header
  - payload – Logs activity, header, and payload (body of the message)



[Assemble an API in the API Designer](#)

© Copyright IBM Corporation 2016

Figure 7-16. Activity log policy

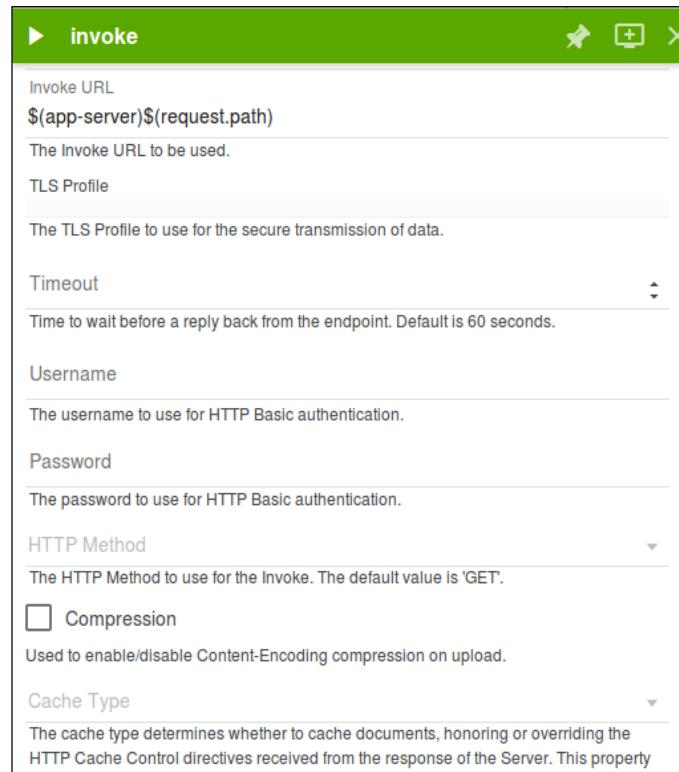
Use the activity-log policy to configure your logging preferences for the API activity that is stored in Analytics.

The Content field is the only required field and it defines the type of content to be logged when the operation is successful.

Restriction: The activity-log policy can be used only with the DataPower Gateway.

## Invoke policy

- Apply the invoke policy to call an existing REST or SOAP service from your assembly
- Uses JSON or XML data
- URL specifies a URL for the target service
- HTTP method. Valid values:
  - GET
  - POST
  - PUT
  - DELETE
  - PATCH
  - HEAD
  - OPTIONS



[Assemble an API in the API Designer](#)

© Copyright IBM Corporation 2016

Figure 7-17. *Invoke policy*

Required fields are title, URL, timeout, and HTTP method.

Default values:

Title – invoke

URL – For a SOAP API, the URL is added by default.

Timeout – 60 seconds

HTTP method – GET

Refer to the Knowledge Center for API Connect V5 for information about the properties of the invoke policy.



## Gateway script policy

```

1 // Require API Connect Functions
2 var apic = require('local:///isp/policy/apim.custom.js');
3
4
5 // Save the Google Geocode response body to variable
6 var mapsApiRsp = apic.getvariable('google_geocode_response
    .body');
7
8
9 // Get location attributes from geocode response body
10 var location = mapsApiRsp.results[0].geometry.location;
11
12
13 // Set up the response data object, concat the latitude and
    longitude
14 var rspObj = {
15
16   "google_maps_link": "https://www.google.com/maps?q="
17   + location.lat + "," + location.lng
18 };
19
20

```

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-18. Gateway script policy

Use the `gatewayscript` policy to execute a specified DataPower `gatewayscript` program.

Gateway Script allows you to write message processing policies for a DataPower device in ECMAScript instead of XSLT that is used in early versions of the DataPower appliance.

Properties of the `gatewayscript` policy:

Title – type string, not required

Description – type string, not required

Source – type string, required.

The variable named `apim` provides the gateway script with access to the DataPower Gateway module.

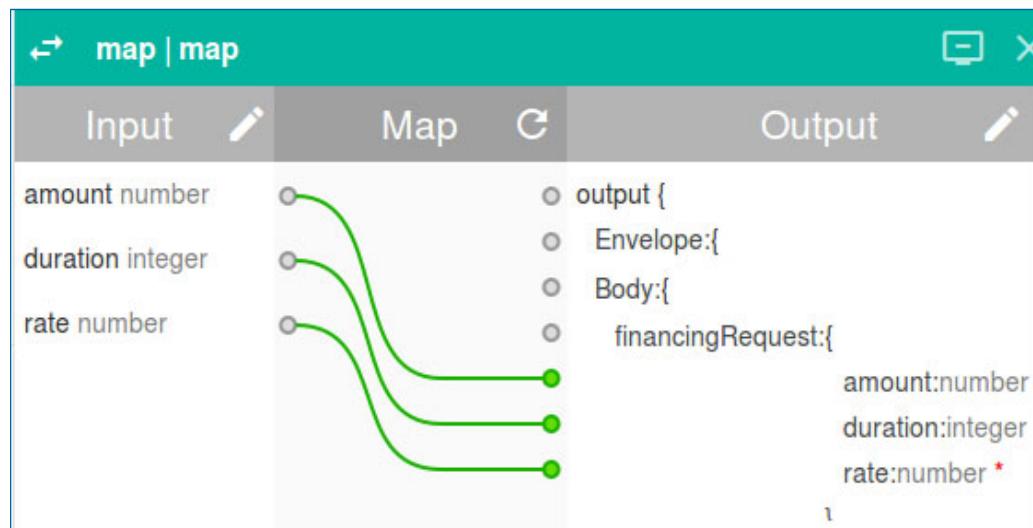
The example below returns a synchronous read of the original request body:

```
var json = apim.getvariable('request.body')
```

Restriction: The `gatewayscript` policy can be used only with the DataPower Gateway.

## Gateway mapping policy

- Use the map policy to apply transformations to your assembly flow and specify mapping between variables



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-19. Gateway mapping policy

Use the map policy to apply transformations to your assembly flow and map between variables.

The example shows a mapping from the input map variables of a REST-type message to the output map variables of a SOAP-type message.

The map policy copies, transforms, or aggregates fields from an input to an output message.

For example, you can combine the results from two messages into one output message. You can concatenate fields, and copy the values of properties from one message to another.

Restriction: The map policy can be used only with the DataPower Gateway.

## 7.2. Assemble an API operation for an existing service

## Assemble an API operation for an existing service

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

*Figure 7-20. Assemble an API operation for an existing service*

## Topics

- API policies

 Assemble an API operation for an existing service

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-21. Topics

## Instructor demonstration (1 of 2)



- In this scenario, you map an existing system API to your API definition with the API Designer.
1. Create an API with LoopBack
  2. Edit the API with the API Designer
  3. Review the draft API that is created
  4. Add the API schema definition for the API
  5. Configure the schema definition properties
  6. Define the API paths
  7. Specify the API parameters
  8. Define the responses for the API operations
  9. Add an operation switch in the assembly
  10. Configure case 0 in the operation switch
  11. Configure case 1 in the operation switch
  12. Define invoke actions in the assembly

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-22. Instructor demonstration (1 of 2)

The steps that are described assume that you have the API Designer toolkit installed either locally or on the course image. You must also have Internet access to the existing system API endpoint at <https://accounts.sample.mybluemix.net/accounts>.

Test that you can access the endpoint before starting this demonstration.

## Instructor demonstration (2 of 2)

- In this scenario, you map an existing system API to your API definition with the API Designer.



**13.** Map the case 0 path to an existing API operation

**14.** Map the case 1 path to an existing API operation

**15.** Start the application and Micro Gateway

**16.** Test the API operation with API Explorer

Figure 7-23. Instructor demonstration (2 of 2)

The steps that are described assume that you have the API Designer toolkit installed either locally or on the course image. You must also have Internet access to the existing system API endpoint at <https://accounts.sample.mybluemix.net/accounts>.

Test that you can access the endpoint before starting this demonstration.

## Step 1: Create an API with Loopback

1. Create an empty LoopBack application with the API Designer

```
$ apic loopback accounts
```

2. In the create Loopback application, accept the default name and directory:

What's the name of your application? (accounts)

Name of the directory to contain the project: (accounts)

Use the arrow on your keyboard to change the application to:

> empty-server (An empty LoopBack API)

3. Change to the accounts directory

```
$ cd accounts
```

*Figure 7-24. Step 1: Create an API with Loopback*

In the first step, open the terminal on Linux and Mac OS operating systems, or the command prompt on Windows operating systems.

If running the demonstration on the course image, create a **projects** directory, if not already created, and change directory to the projects directory.

Type the command **apic loopback accounts** in the prompt.

Accept the default name for the application and project directory (accounts).

Select empty-server for the response to the question what kind of application do you have in mind.

Ensure that an application named accounts is generated without errors into the accounts folder.

Change directory to the newly-created accounts directory.

## Step 2: Edit the API

- Start the API Designer application

```
$ apic edit
```

The API Designer starts and displays the API in the browser

- Click the **accounts** link in the APIs tab to configure the API

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-25. Step 2: Edit the API

Type the command **apic edit** in the terminal window.



### Note

If the API Designer is installed on your own workstation, you can skip the prompt to sign on to Bluemix by typing:

**SKIP\_LOGIN=true apic edit**



## Step 3: Review the draft API

- LoopBack creates a Swagger file that represents your draft API. Use the **API Editor** to review and define your API details.

The screenshot shows the IBM API Connect interface with the 'accounts' API selected. The 'Design' tab is active. The left sidebar lists categories like Host, Base Path, Schemes, Consumes, Produces, Lifecycle, Policy Assembly, Security Definitions, Security, Properties, and Paths. The main content area shows the API's title as 'accounts', name as 'accounts', and version as '1.0.0'. A scroll bar is visible on the right side of the main content area.

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

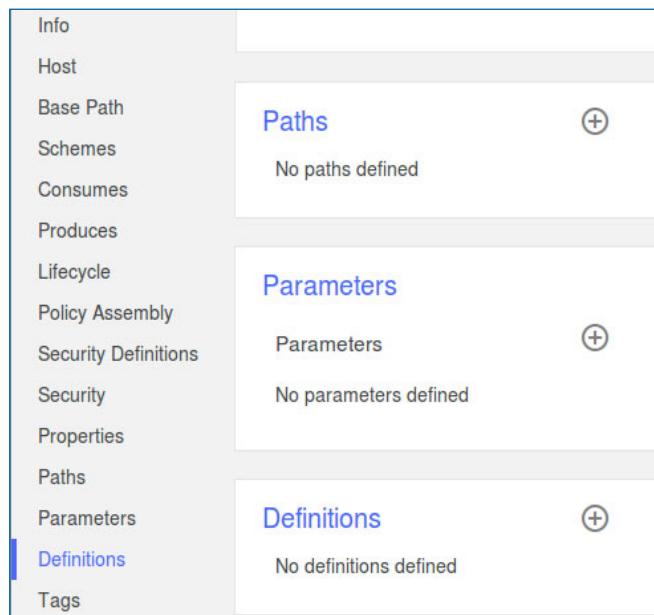
Figure 7-26. Step 3: Review the draft API

The **API Designer** is a web application form that edits your application's API definition. The Designer is part of the **API Editor** web application.

- The **API definition** specifies the operations, message formats, processing policies in your API. API Connect uses the Swagger V2.0 document format for the API Definition. In this leftmost column, the API Designer view lists the API definition categories.
- To open the API Designer, create an API or open an existing API in the API Editor. Select the **Design** tab.
- You can also edit the source version of the API definition in a built-in text editor, in the **Swagger YAML** (Yet another markup language) format.
- In this example, you see that the title, name, and version number for your API are already set.
- After you make any changes to the file later, you select **Save** to commit your changes.

## Step 4: Add the schema definition for the API

1. Select **Definitions** in the navigation pane
2. Add a definition to your API



The screenshot shows the 'Definitions' section selected in the navigation pane on the left. The main area displays three categories: 'Paths', 'Parameters', and 'Definitions'. Each category has a '+ icon' to add new items. Below each category, it says 'No paths defined', 'No parameters defined', and 'No definitions defined' respectively.

Figure 7-27. Step 4: Add the schema definition for the API

Select **Definitions** in the navigation panel. Then, click the icon to add a definition.

## Step 5: Configure the schema definition properties

1. Type **account** in the Name field
2. Add the property fields **id**, **name**, and **email**

Properties					Add Property
*	Property Name	Description	Type	Example	Actions
<input checked="" type="checkbox"/>	id		string		<>
<input type="checkbox"/>	name		string		<>
<input type="checkbox"/>	email		string		<>

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-28. Step 5: Configure the schema definition properties

Click the new-definition-1 link.

In the Definitions dialog, type **account** in the Name field. Leave the Type field as object.

Click Add Property twice, then add the fields with the following properties:

Property Name Type Required

id string checked

name string

email string

## Step 6: Define the API paths

1. Select **Paths**.
2. Add a path named `/accounts` to your API
3. Add a second path named `/accounts/{accountId}`

The screenshot shows the 'Paths' section of the API Designer. There are two entries:

- Path:** `/accounts/{accountId}`. It includes an 'Add Operation' button and a trash icon.
- Path:** `/accounts`. It includes an 'Add Operation' button and a trash icon.

Under each path, there is a table for operations:

Method	Path	Action
GET	<code>/accounts/{accountId}</code>	trash icon
GET	<code>/accounts</code>	trash icon

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-29. Step 6: Define the API paths

The **Paths** in your API definition represent a route to an operation.

Add the path named `/accounts` and a second path named `/accounts/{accountId}`.

The GET operation is added automatically for each path that you add.

To call an API operation, the API consumer makes an HTTP request with a specific HTTP method and URL path. In this example, the GET `/accounts/{accountId}` API operation returns the account details for the customer with the specified identifier.

## Step 7: Specify the API parameters

1. Click **Add Parameter** in the /accounts/{accountId} path
2. Select Add New Parameter
3. Change the name to accountId. Then, select **Path** in the Located In field. Leave the remaining fields as default (Required and string)
4. Save the changes

The screenshot shows the 'API Designer' interface. At the top, there's a 'Path' input field containing '/accounts/{accountId}' with an 'Add Operation' button and a trash icon to its right. Below this is a 'Parameters' section with a table:

Name	Located In	Description	Required	Type
accountId	Path		<input checked="" type="checkbox"/>	string

At the bottom of the interface, there's a 'GET' method button followed by the path '/accounts/{accountId}' and a trash icon.

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-30. Step 7: Specify the API parameters

Add a parameter for the GET /accounts/{accountId}.

Name  
accountid

Located In  
Path

Required  
checked

Type  
string

Save the changes.



### Information

The warning that is displayed when you click the Validation icon tells you that you have defined a schema, but that it is not used.

You fix that next.

## Step 8: Define responses for the API operations

1. Click GET /accounts/{accountId}
2. In the Responses area, for Status Code 200, change the Description to Successful request and select **accounts** in the schema drop-down

Responses		
Status Code	Description	Schema
200	Successful request	account ▾

3. Click GET /accounts
4. In the Responses area, for Status Code 200, change the Description to Successful request and select **accounts** in the schema drop-down
5. Save all changes

[Assemble an API in the API Designer](#)

© Copyright IBM Corporation 2016

Figure 7-31. Step 8: Define responses for the API operations

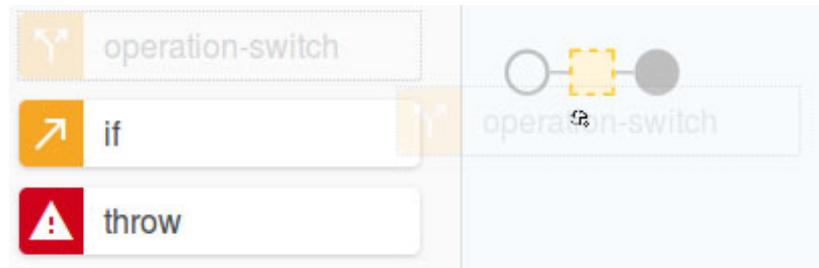
Edit the GET /accounts/{accountId} and go to the responses area. Change the description for the status code 200 to "Successful request". Select account in the drop-down list for the schema.

Perform the same steps for the GET /accounts operation.

Save your changes.

## Step 9: Add an operation-switch in the assembly

1. Click the Assemble tab in API Designer
2. Delete the existing Invoke icon
3. From the palette, select an **operation-switch** and drop it between the start and end icons in the flow diagram



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-32. Step 9: Add an operation-switch in the assembly

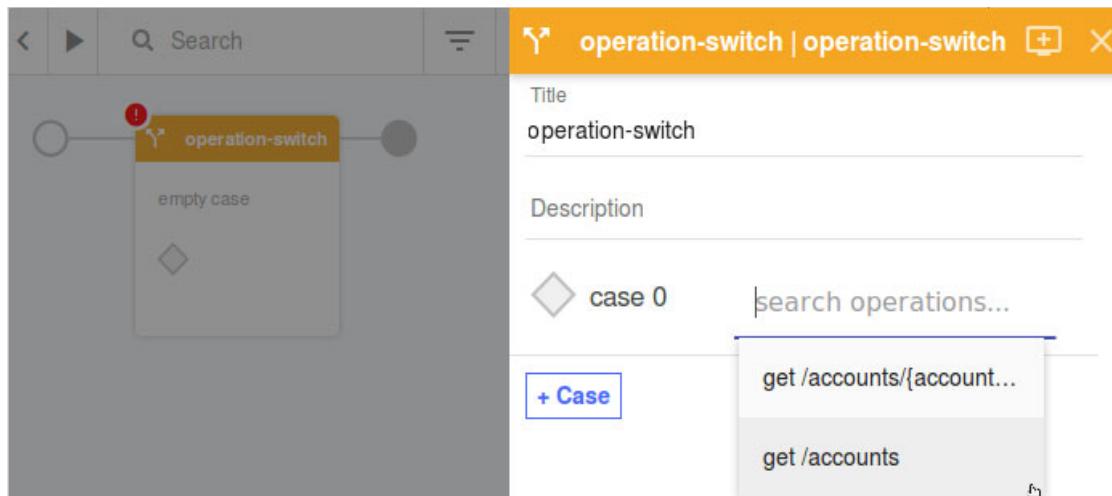
Open the Assembly editor by clicking the Assemble tab in the API Designer.

Delete the Invoke action from the flow diagram.

Drag and operation-switch from the palette and drop it on the flow diagram.

## Step 10: Configure case 0 in the operation-switch

1. Click the **operation-switch** in the flow diagram to open the Properties dialog
2. Select `get /accounts` from the search operations in the Properties



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-33. Step 10: Configure case 0 in the operation-switch

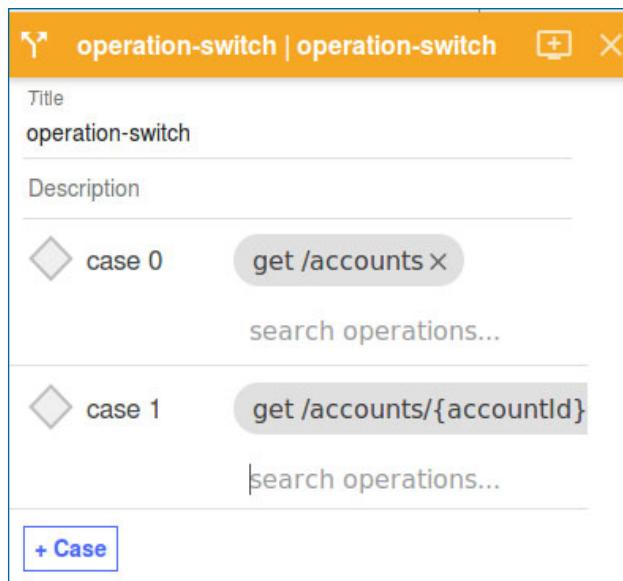
Click operation-switch in the flow diagram.

The Properties dialog opens.

Select `get /accounts` from the search operations drop-down list.

## Step 11: Configure case 1 in the operation-switch

1. Click the **+ Case** in the Properties dialog
2. Select `get /accounts/{accountId}` from the search operations in the Properties



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-34. Step 11: Configure case 1 in the operation-switch

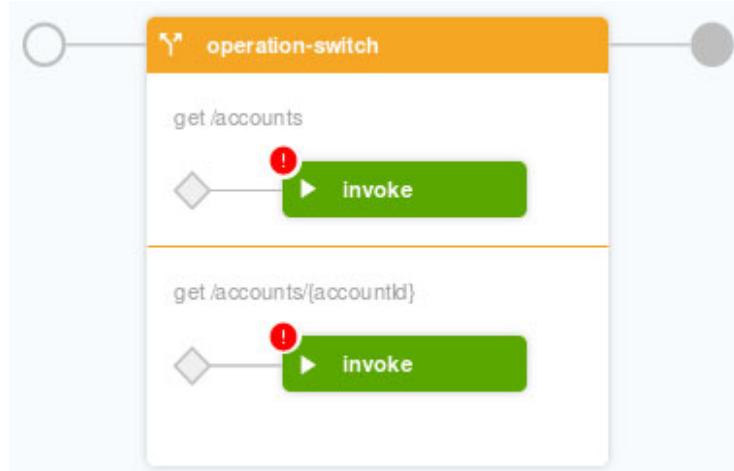
Click **+ Case** in the Properties dialog to add a case.

Select `get /accounts/{accountId}` from the search operations drop-down for case 1.

Close the Properties dialog.

## Step 12: Define invoke actions in the assembly

1. From the palette, select an **invoke** action and drop it to the right of the case 0 icon
2. From the palette, select an **invoke** action and drop it to the right of the case 1 icon



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

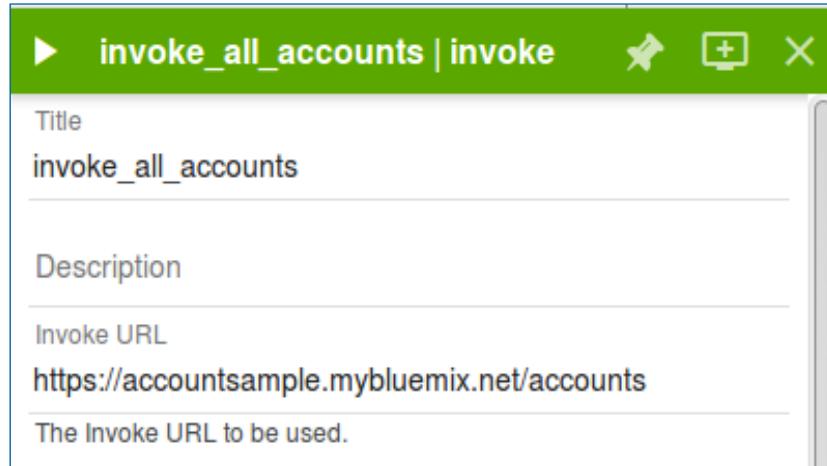
Figure 7-35. Step 12: Define invoke actions in the assembly

From the palette, select an **invoke** action and drop it to the right of the case 0 icon.

Repeat the process to drop another invoke action to the right of the case 1 icon.

## Step 13: Map the case 0 path to an existing API operation

1. Click the first Invoke action to open the Properties dialog
2. Type: invoke\_all\_accounts in the Title field
3. Type: https://accounts.sample.mybluemix.net/accounts in the Invoke URL field



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

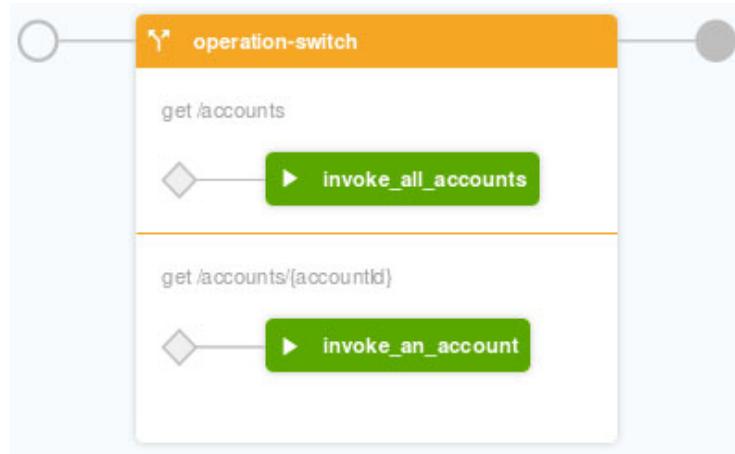
Figure 7-36. Step 13: Map the case 0 path to an existing API operation

In the **Assemble** flow view, click the first **invoke** action to open the Properties dialog. The invoke action reads the API operation request and forwards the request to the API endpoint specified by the Invoke URL variable. In this scenario, you set the invoke URL to the existing API operation. The **https://accounts.sample.mybluemix.net/accounts** API endpoint represents your existing API operation.

Close the Properties dialog.

## Step 14: Map the case 1 path to an existing API operation

1. Click the second Invoke action to open the Properties dialog
2. Type: `invoke_an_account` in the Title field
3. In the Invoke URL field, type:  
`https://accounts-sample.mybluemix.net/accounts/${request.parameters.accountId}`
4. The flow diagram is complete. Save the changes.



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-37. Step 14: Map the case 1 path to an existing API operation

In the **Assemble** flow view, click the first **invoke** action to open the Properties dialog. The invoke action reads the API operation request and forwards the request to the API endpoint specified by the Invoke URL variable. In this scenario, you set the invoke URL to the existing API operation. The **`https://accounts-sample.mybluemix.net/accounts/${request.parameters.accountId}`** API endpoint represents your existing API operation.

Close the Properties dialog.

Save the changes.

Both of the invoke actions are displayed in the flow diagram.



## Step 15: Start the application and Micro Gateway

1. Start the application and the Micro Gateway
  - Click **Run**, then **Start**
  - The application and the Micro Gateway are started

A screenshot of the API Designer interface. At the top, there is a navigation bar with five items: "Products", "APIs", "Models", "Data Sources", and "Run". The "Run" button is highlighted with a blue underline. Below the navigation bar, the word "Running" is displayed in green. Underneath "Running", there are two lines of text: "Application: http://127.0.0.1:4001/" and "Micro Gateway: https://127.0.0.1:4002/".

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-38. Step 15: Start the application and Micro Gateway

Start the application and the Micro Gateway.

In the API Designer, click **Run**. Then, click **Start**.

## Step 15: Test the API operation with API Explorer

1. Click **Explore**
2. Click the GET /accounts/{accountId} API operation in the API Explorer

The screenshot illustrates the API Explorer interface in the IBM API Designer. The left sidebar lists API operations under 'accounts 1.0.0': 'GET /accounts/{accountId}' is selected and highlighted with a red box and a yellow circle labeled '2'. The middle section shows the details for this selected operation, including its path ('GET /accounts/{accountId}') and parameters ('accountId'). The right section displays sample code for calling the API in various programming languages, specifically Node.js, with a curl command. Numbered callouts point to specific elements: '1' points to the 'Node' tab in the language selector; '2' points to the selected API operation in the left sidebar; '3' points to the 'Schema' link in the responses table; and '4' points to the sample code in the right panel.

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-39. Step 15: Test the API operation with API Explorer

The API Explorer page is a web-based test client for your API operations.

1. To open the API Explorer, select **Explore** from the API Designer navigation bar.
2. The leftmost column displays the API operation that you defined in the product.
3. The middle column displays the properties of the API operation that you selected. In this example, the GET /accounts/{accountId} API operation takes one parameter, the account identifier.
4. The rightmost column displays sample code to call the API operation in several programming languages.

To open the test client, scroll down on the rightmost column.

## Example: Test an invoke operation in the micro gateway

The screenshot shows the IBM API Designer interface. On the left, the 'Test Client' panel displays the URL `https://localhost:4002/api/accounts/{accountId}`. It includes sections for 'Identification' (Client ID: default, Client secret: SECRET), 'Content-Type' (application/json), 'Accept' (application/json), and 'Parameters' (accountId: 10). A 'Generate' button is also present. A 'Call operation' button is highlighted with a red box and a yellow arrow labeled '6'. On the right, the 'Call operation' panel shows the request details: GET `https://localhost:4002/api/accounts/10`, APIM-Debug: true, Content-Type: application/json, Accept: application/json, X-IBM-Client-Id: default, and X-IBM-Client-Secret: SECRET. The response section shows a status code of 200 OK, headers including content-type: application/json; charset=utf-8, x-global-transaction-id: 2928084111, x-ratelim-limt: 100, x-ratelim-remaing: 99, and x-ratelim-reset: 3599999. The response body is a JSON object:

```
{
  "id": "10",
  "name": "Isabella Martinez",
  "email": "isabella.martinez@example.com"
}
```

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-40. Example: Test an invoke operation in the micro gateway

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

5. The **parameters** section lists the required and optional values that the API operation expects. In this example, you enter an accountId with a value between 1 and 10.
6. Select **Call operation** to make an HTTP or HTTPS request to the API operation.
7. Scroll down further to review the request and response messages from the API call.
8. The response message header displays the HTTP status code and response header values.
9. The last section displays the HTTP response message body. In this example, the API Gateway forwarded the request to the accountsample.mybluemix.net server. The existing accounts API on the server returned a customer account record in JSON format.

Repeat the test for the GET /accounts operation.

## Unit summary

- Describe the concept of API policies
- Identify the types of policies
- Describe the capabilities of the Micro Gateway policies
- Describe the capabilities of the DataPower gateway policies
- Describe how to add components in an assembly
- Describe the activity log policy
- Describe the invoke policy
- Describe the gateway script policy
- Describe the map policy
- Create an assembly for an API operation that calls an existing service

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

*Figure 7-41. Unit summary*

## Review questions

1. True or False: There is no difference between the Micro Gateway javascript policy and the DataPower gatewayscript policy.
  
2. Which of these statements apply to user-defined policies?
  - A. Defined externally from API Connect
  - B. Imported into an API Connect catalog
  - C. Policy becomes available to be placed into an assembly in the Assembly editor
  - D. All the above.



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-42. Review questions

Write your answers here:

- 1.
- 2.

## Review answers

1. True or False: There is no difference between the Micro Gateway javascript policy and the DataPower gateway script policy.

The answer is False.

The javascript policy is written to run on the Node.js Micro Gateway, while the gatewayscript policy runs on DataPower



2. Which of these statements apply to user-defined policies?

- A. Defined externally from API Connect
- B. Imported into an API Connect catalog
- C. Policy becomes available to be placed into an assembly in the Assembly editor

D. All the above.

The answer is D.

## Exercise: Advanced API Assembly

Lab 5

Assemble an API in the API Designer

© Copyright IBM Corporation 2016

*Figure 7-44. Exercise: Advanced API Assembly*

## Exercise objectives

- Create a new API, including object definitions and paths
- Configure an API to access an existing SOAP service
- Use the source editor to import an existing API definition
- Map data retrieved from multiple API calls into an aggregate response
- Use gatewayscript directly within an API assembly



Assemble an API in the API Designer

© Copyright IBM Corporation 2016

Figure 7-45. Exercise objectives

---

# Unit 8. Publish APIs

## Estimated time

00:45

## Overview

In this unit, you examine how to publish an API with plans and products. You learn how to make APIs available to application developers by publishing products to the Developer Portal.

## How you will check your progress

- Review questions
- Lab exercise

## Unit objectives

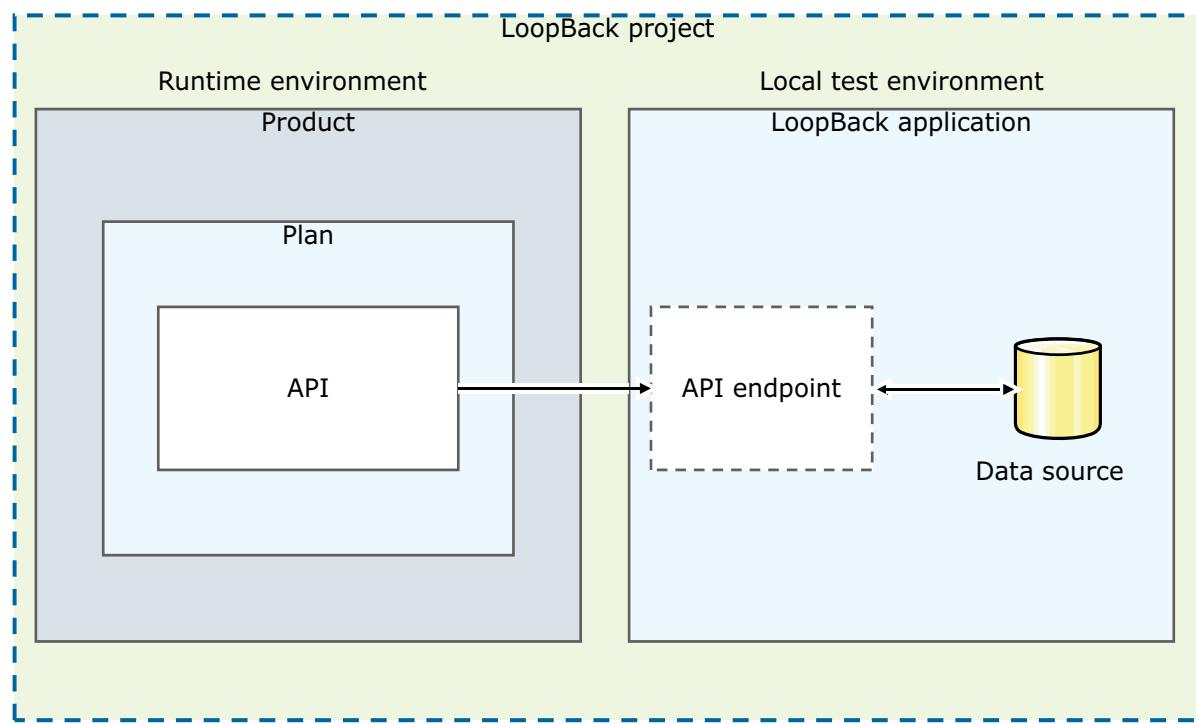
- Explain the purpose of plans and products
- Make a service available to consumers with products and plans
- Explain the concept of catalogs in API Connect
- Stage and publish a product
- Manage your products in a catalog

Publish APIs

© Copyright IBM Corporation 2016

*Figure 8-1. Unit objectives*

## Product, plan, API hierarchy



Publish APIs

© Copyright IBM Corporation 2016

Figure 8-2. Product, plan, API hierarchy

Products, plans, and APIs are all represented in a YAML file within a LoopBack project.

Testing on the workstation:

You can test your API endpoints from the API Designer explorer by clicking the **Run** followed by the **Start** option.

After the server and Micro Gateway are both started, click Explore in the API Designer.

Select the API REST operation that you want to test. Then, use the cURL command to run the operation.

Testing in the runtime environment:

API endpoints can also be tested from the Developer Portal after the API and its containing Product are published. Sign on to the Developer Portal in a browser. Select the API operation that you want to test. Then, use the cURL command to invoke the published endpoint.

## 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

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-3. 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 levels of performance.

## Product YAML file

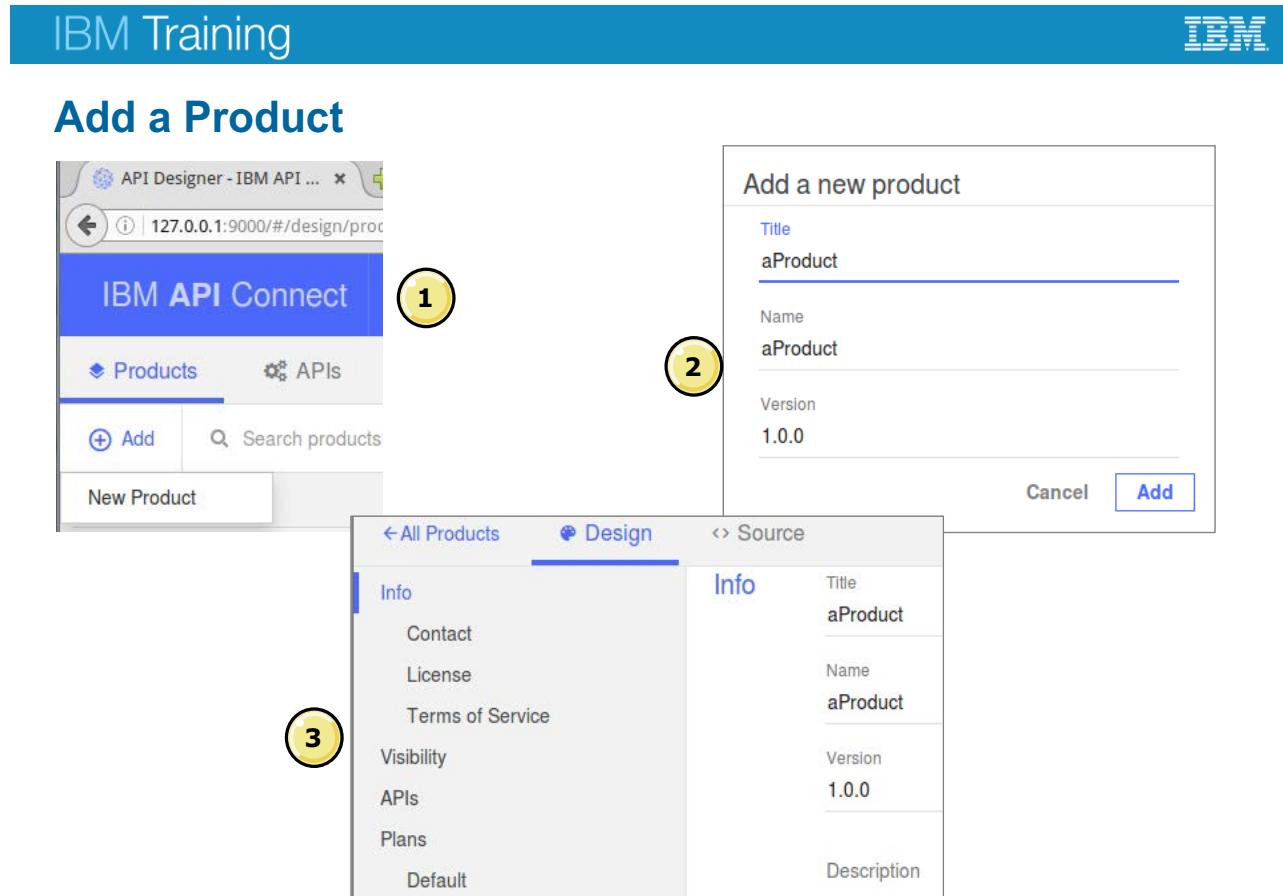
```
product: '1.0.0'
info:
  name: hello-world
  title: hello-world
  version: 1.0.0
apis:
  'hello-world':
    $ref: hello-world.yaml
visibility:
  view:
    type: public
  subscribe:
    type: authenticated
plans:
  default:
    title: Default Plan
    description: Default Plan
    approval: false
    rate-limit:
      value: 100/hour
      hard-limit: false
```

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-4. Product YAML file

Products can be represented by using a YAML file in a similar fashion to how APIs can be represented by using Swagger.



Publish APIs

© Copyright IBM Corporation 2016

Figure 8-5. Add a Product

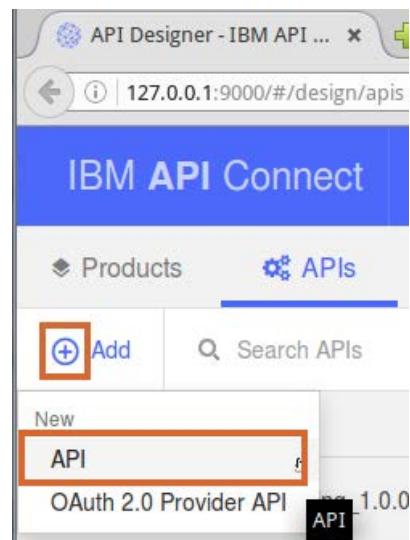
Type **apic edit** to open the API Designer.

1. Select the Products tab, then click the Add icon.
2. Type a title, name, and version for the product in the Add a new product dialog. 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
  - Generate a LoopBack application



Publish APIs

© Copyright IBM Corporation 2016

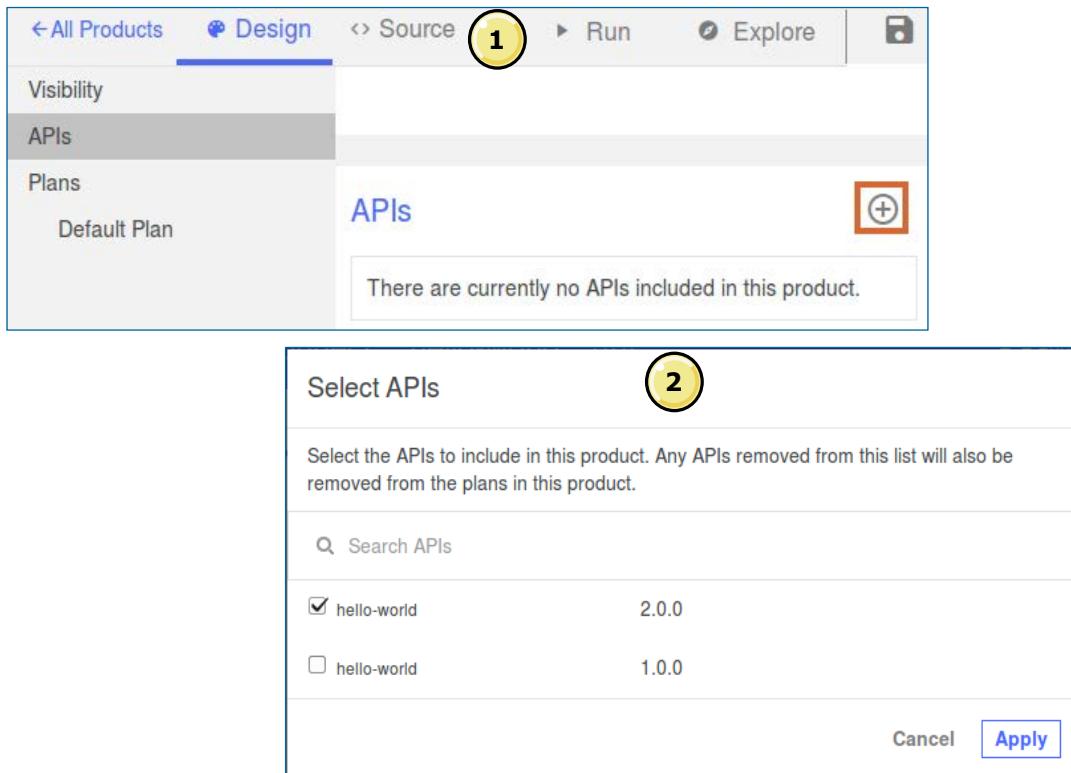
Figure 8-6. Add an API

You have the option to add an API to a product when creating an API with the API Designer, or you can add APIs to a product later.

# 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 yellow circle containing '1'), 'Source', 'Run', 'Explore', and a save icon. On the left, a sidebar has tabs for 'Visibility', 'APIs' (highlighted with a yellow circle containing '1'), and 'Plans'. Under 'Plans', there's a 'Default Plan'. In the main area, under 'APIs', it says 'APIs' and there's a red-bordered '+' button. Below that, a message says 'There are currently no APIs included in this product.'

**Select APIs** (highlighted with a yellow circle containing '2')

Select the APIs to include in this product. Any APIs removed from this list will also be removed from the plans in this product.

Search APIs:

<input checked="" type="checkbox"/> hello-world	2.0.0
<input type="checkbox"/> hello-world	1.0.0

Cancel **Apply**

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-7. Add existing APIs to a Product

In the API Designer, click the APIs tab.

Click the Add icon. Select from the list of available APIs.

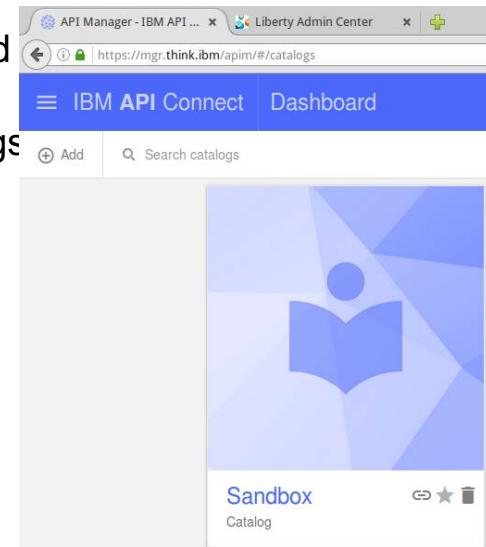
Then, click Apply.

Save the Product.



## Catalogs

- Catalogs are useful for separating Products and APIs for testing and production
- The URL 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



Publish APIs

© Copyright IBM Corporation 2016

*Figure 8-8. Catalogs*

Products must be staged and published to a catalog to become available to application developers.

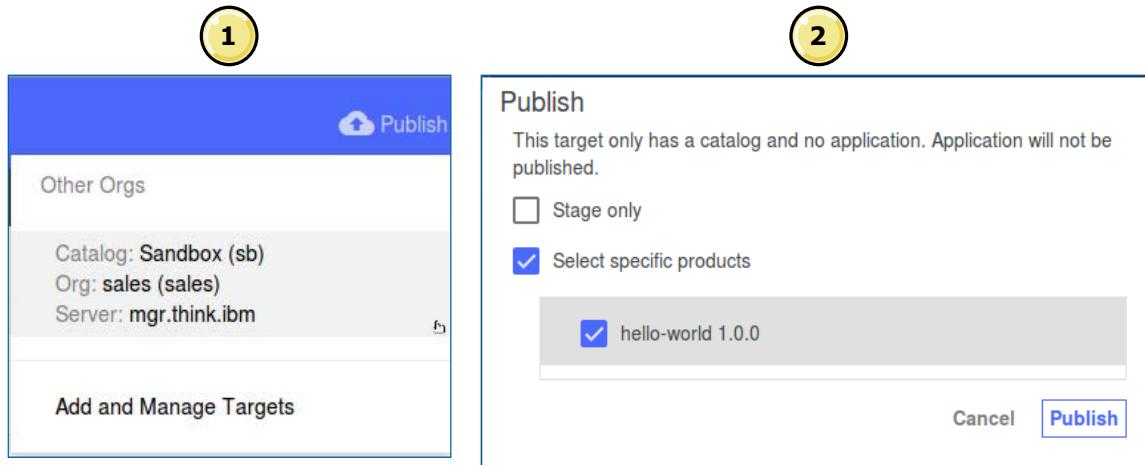
In a typical configuration, an API provider organization uses a Sandbox catalog for testing APIs under development and a production catalog for hosting APIs that are ready for full use.

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



Publish APIs

© Copyright IBM Corporation 2016

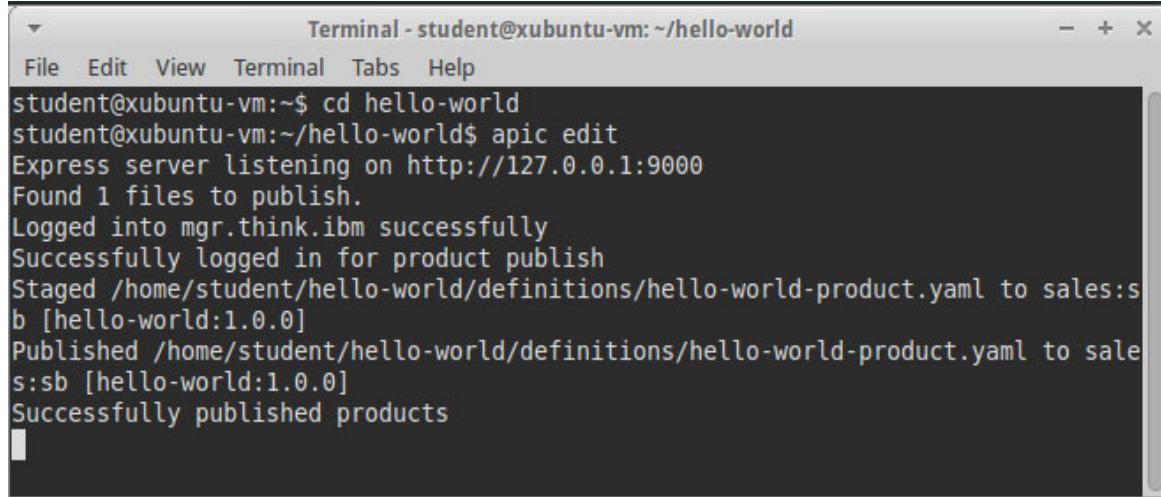
Figure 8-9. Publish a plan and API

## Publish a plan and API result

- Successful 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 8-10. 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
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]
```

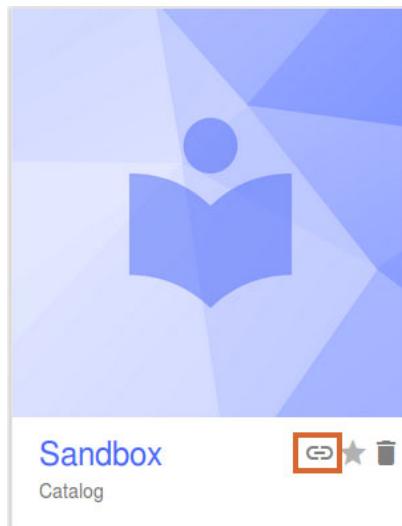
## Successfully published products

---



## 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

Publish APIs

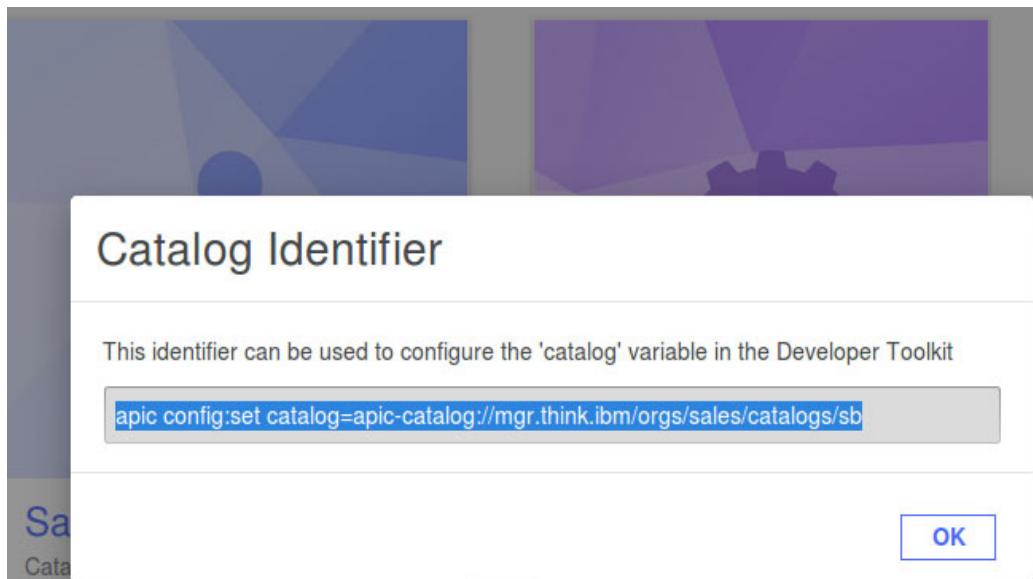
© Copyright IBM Corporation 2016

Figure 8-11. Publish from the terminal: preparation (1 of 2)



## Publish from the terminal: preparation (2 of 2)

- Copy the catalog identifier string to a clipboard



Publish APIs

© Copyright IBM Corporation 2016

Figure 8-12. 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 8-13. Publish a LoopBack project from the terminal*

1. Sign on to API Manager, if you are not 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, and

*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 Product in API Manager

- Published Products can be viewed and managed in API Manager

A screenshot of the IBM API Connect Sandbox interface. The browser title bar says "API Manager - IBM API ...". The address bar shows the URL "https://mgr.think.ibm/apim/#/catalogs/5730eb7fe4b0a6517448ed8d/products". The main header has tabs for "IBM API Connect" and "Sandbox". Below the header, there are navigation links: "All Catalogs", "Products" (which is selected), "Approvals", "Subscriptions", and "Settings". A search bar says "Search products". The main content area displays a table with two columns: "Title" and "State". There is one row visible: "hello-world hello-world:1.0.0" under "Title" and "Published" under "State". To the right of the "Published" state is a small red-bordered icon with a gear and a plus sign. Below the table is a "Manage" button. At the bottom of the page, there are links for "Publish APIs" and "© Copyright IBM Corporation 2016".

Title	State
hello-world hello-world:1.0.0	Published

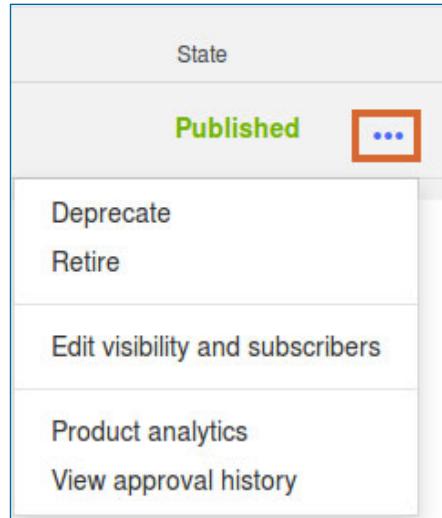
Publish APIs

© Copyright IBM Corporation 2016

Figure 8-14. Manage published Product in API Manager

## Options on the Manage menu for a published Product

- Manage lifecycle and subscription options from the Manage menu



Publish APIs

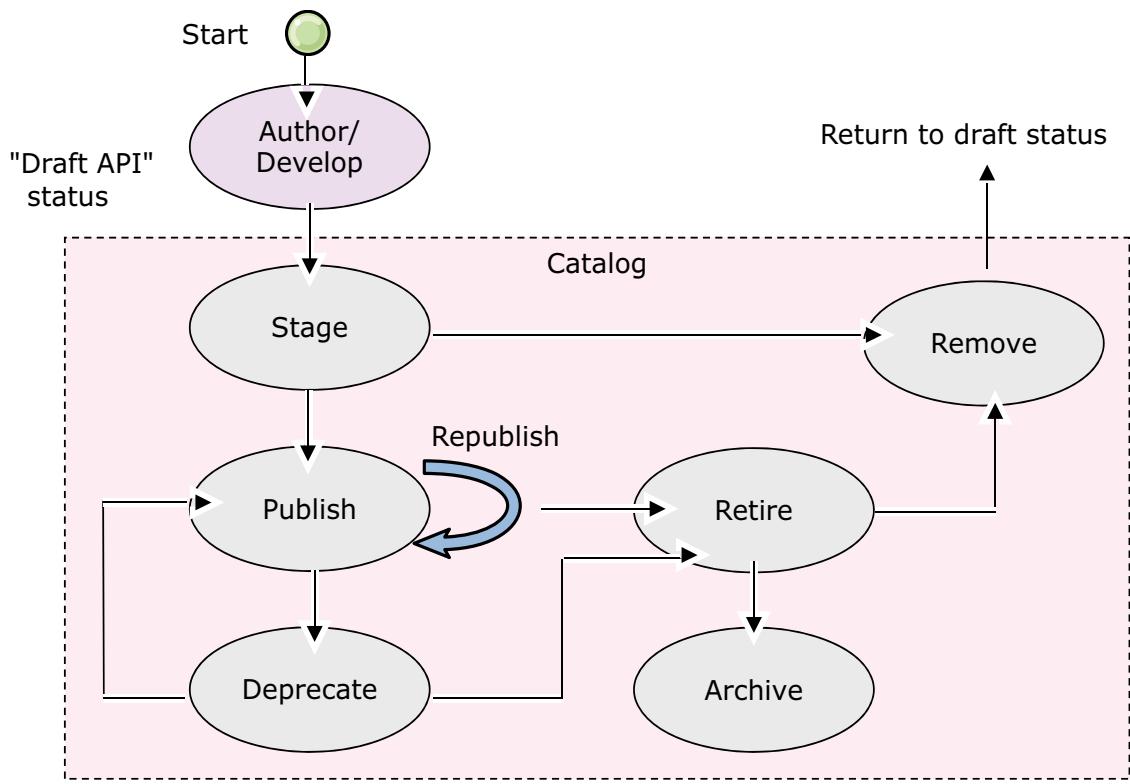
© Copyright IBM Corporation 2016

Figure 8-15. 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 viewing product analytics and approval history.

## Lifecycle of Products and API resources



Publish APIs

© Copyright IBM Corporation 2016

*Figure 8-16. 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 requisite 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 can not be viewed or subscribed to, and all of the associated APIs are stopped.

Product versions in the archived state are similar to those 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

Edit visibility and subscribers

hello-world	<input type="checkbox"/> Temporarily disable visibility	<input type="checkbox"/> Temporarily disable subscribability
Visible to:	<a href="#">Public (Developer Portal)</a> ▾	
All developers will be able to see this product	All authenticated developers in consumer organizations who have signed up for this developer portal can see this product	
Changes made here will not affect existing subscriptions		
<a href="#">Republish</a>		<a href="#">Cancel</a>

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-17. Republish a Product version

You can change the visibility and subscription options for a previously published Product from the Edit visibility and subscribers dialog.

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 8-18. 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 dialog 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 will not be 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 if *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 if the APIs are being used in the Developer Portal

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-19. Stage and publish a previously published Product (1 of 2)

If you publish a Product to a Sandbox catalog, editing it through the Products tab of API Designer or API Manager will enable you to re-stage 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 recommended that when you stage a Product, you then **create a new version** of the Product in the API Designer to edit in future

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-20. 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 recommended that you then create a new version of the Product to edit in future, so as to avoid confusion regarding the properties of the published Product.



## Create a version of a Product in the API Designer

The screenshot shows the IBM API Connect interface with the following steps highlighted:

- Step 1:** The "Products" tab is selected in the top navigation bar.
- Step 2:** A specific product, "hello-world", is selected.
- Step 3:** The "More Actions" icon (three dots) is clicked, revealing a context menu with "Save as a new version" and "Delete".
- Step 4:** The "Save as a new version" dialog box is open, showing fields for "Version" (set to 2.0.0) and "File name" (set to hello-world). The "Save as a new version" button is highlighted.

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-21. Create a version of a Product in the API Designer

In API Designer, you can have multiple versions of a Product.

To create a new Product version:

1. Select the Products tab in API Designer. The Products tab opens.
2. Click the Product you want to create your new 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

A screenshot of the IBM API Connect API Designer interface. The title bar shows "IBM API Connect" and "hello-world 2.0.0". The navigation bar has links for "All Products", "Design" (which is selected), and "Source". On the left, a sidebar menu includes "Info", "Contact", "License", "Terms of Service", "Visibility", "APIs", "Plans", and "Default Plan". The main content area is titled "Info" and displays the following details:

Title	hello-world
Name	hello-world
Version	2.0.0
Description	[Empty]

Publish APIs

© Copyright IBM Corporation 2016

Figure 8-22. 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 if desired.



## Permissions for managing catalogs

- By default, all roles can stage, view, and manage Products in the Sandbox catalog

The screenshot shows the IBM API Connect interface with the title bar "IBM API Connect" and "Sandbox". The top navigation bar includes links for "All Catalogs", "Products", "Approvals", "Subscriptions", "Developers", "Analytics", and "Settings". The "Settings" link is underlined, indicating it is selected. Below the navigation is a secondary navigation bar with tabs: "Configuration", "Portal", "Permissions" (which is highlighted), and "Policies". A message "Owner has all catalog permissions" is displayed above the permission table. The table lists three permissions: Stage, View, and Manage, each associated with four roles: Publisher, API Developer, Administrator, and Product Manager. All four roles have "X" next to them, indicating they have permission.

Permission Description	Role
Stage Allow staging of products to this catalog	Publisher X API Developer X Administrator X Product Manager X
View Allow viewing and listing of products in this catalog	Publisher X API Developer X Administrator X Product Manager X
Manage Allow management of product lifecycle (publishing, deprecating, retiring and archiving)	Publisher X API Developer X Administrator X Product Manager X

[Publish APIs](#)

© Copyright IBM Corporation 2016

Figure 8-23. Permissions for managing catalogs

By default, all users can stage, view, and manage Products in sandbox catalogs.

Only the organization owner has permissions to stage, view and manage Products in a non-sandbox catalog.

You can customize the permissions for a non-sandbox catalog in the permissions page.

## Unit summary

- Explain the purpose of plans and products
- Make a service available to consumers with products and plans
- Explain the concept of catalogs in API Connect
- Stage and publish a product
- Manage your products in a catalog

Publish APIs

© Copyright IBM Corporation 2016

*Figure 8-24. 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 the above.
  
2. Which of these statements is NOT a lifecycle state:
  - A. Stage
  - B. Publish
  - C. Catalog
  - D. Retire



Publish APIs

© Copyright IBM Corporation 2016

Figure 8-25. Review questions

Write your answers here:

- 1.
- 2.
- 3.

## 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 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.



Publish APIs

© Copyright IBM Corporation 2016

Figure 8-26. Review answers

## Exercise: Working with API Products

### Lab 6

Publish APIs

© Copyright IBM Corporation 2016

*Figure 8-27. Exercise: Working with API Products*

## Exercise objectives

- Create a Product
- Attach APIs to a Product
- Create a Plan
- Publish a Product



Publish APIs

© Copyright IBM Corporation 2016

*Figure 8-28. Exercise objectives*

---

# Unit 9. Use an API through the Developer Portal

## Estimated time

01:00

## Overview

In this unit, you review the features of the Developer Portal in API Connect. You learn how to use the developer portal to explore the API resources. Browse and select from available plans. Learn how register applications, bind applications to plans, and test applications from the Developer Portal.

## How you will check your progress

- Review questions
- Lab exercise

## Unit objectives

- Describe how to access the Developer Portal
- Review the differences between the Developer Portal with and without signing on
- Describe how users are added to the Developer Portal
- Describe how to create an application on the Developer Portal
- Illustrate how applications can be subscribed to plans
- Explain how applications are tested on the Developer Portal
- Describe how to add a custom theme to the Developer Portal

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-1. Unit objectives*

## User stories and roles when using APIs

- User story: As an API Developer, you want to view, validate, and test the APIs that are published to the Developer Portal
  - Role: API Developer
- User story: As an Application Developer, you want to discover and test APIs and register applications that use APIs
  - Role: App Developer



[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-2. User stories and roles when using APIs

## 9.1. Browse and sign on to the Developer Portal to explore APIs

**Browse and sign on to the  
Developer Portal to explore  
APIs**

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-3. Browse and sign on to the Developer Portal to explore APIs*

## Topics

 Browse and sign on to the Developer Portal to explore APIs

- Create an application that uses APIs
- Test an API on the Developer Portal
- Customize the Developer Portal

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-4. Topics*



## Accessing the Developer Portal

- Locating the Developer Portal URL in API Manager:
  - The URL or host name for the Developer Portal is displayed on the Portal tab of the Settings view for the target catalog

A screenshot of the IBM API Connect interface. At the top, there's a blue header bar with the text "IBM API Connect" and "Sandbox". Below the header, there's a toolbar with icons for "All Catalogs", a dropdown, a checkmark, a gear, users, and metrics. The main menu has tabs: "Configuration", "Portal" (which is selected and highlighted in blue), "Permissions", and "Policies". Under the "Portal" tab, the section "Portal Configuration" is shown. It contains two radio buttons: "None" (unselected) and "IBM Developer Portal" (selected). Below the radio buttons is a field labeled "URL" containing the value "https://192.168.225.20", which is enclosed in a red rectangular box.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-5. Accessing the Developer Portal

The URL or hostname of the Developer Portal is defined in the portal configuration of API Manager. Developers use this URL or hostname to access the Developer Portal from a browser session. The Developer Portal is a separate appliance in the API Connect solution and must be running for users to access the URL.

The screenshot shows the IBM API Connect Sandbox interface. At the top, there's a blue header bar with the text "IBM Training" on the left and the "IBM" logo on the right. Below the header, a teal banner displays the message "View published products in API Manager". The main content area has a blue header "IBM API Connect Sandbox". A green "Success" message at the top of this area says "think (version 1.0.0) has been published to Sandbox". Below this, there's a search bar and some navigation icons. The main content is a table listing published products:

Title	State	Actions
hello-world hello-world:1.0.0	Published	...
think think:1.0.0	Published	...

For the "think" product, detailed information is shown in a modal window:

- APIs:**
  - financing** 1.0.0: Offline / Online switch is off, Subscribers bar is empty.
  - inventory** 1.0.0: Offline / Online switch is off, Subscribers bar is empty.
  - logistics** 1.0.0: Offline / Online switch is off, Subscribers bar is empty.
  - oauth** 1.0.0: Offline / Online switch is off, Subscribers bar is empty.
- Plans:**
  - Silver**: 0 Subscribers, Subscribers bar is empty.
  - Gold**: 0 Subscribers, Subscribers bar is empty.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-6. View published products in API Manager

For the Sandbox catalog, all users can view and manage the list of published products in API Manager.

As an API Developer, I want to see which Products and APIs are published.



## Developer Portal: public interface

- APIs that are published with public visibility are displayed

A screenshot of a web browser displaying the IBM API Connect /dev public interface. The URL in the address bar is https://192.168.225.20. The page has a blue header with the text "Innovate with our APIs". Below the header, there is a welcome message: "Welcome to our API portal where you will find a great selection of APIs for your awesome innovative apps". A section titled "Popular APIs" is shown, featuring a green circular icon with three white layers, representing the "hello-world" API version 1.0.0. The browser's navigation bar and various menu options like Home, Getting Started, API Products, Blogs, Forums, and Support are visible at the top.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-7. Developer Portal: public interface

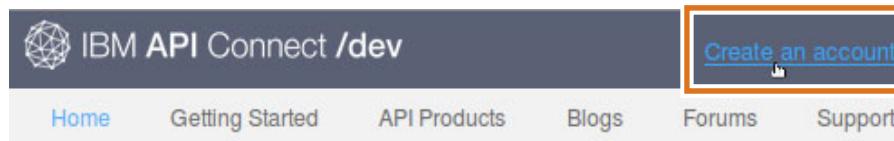
Although many products and APIs might be published to the Developer Portal, only APIs that are published with a visibility of **public** are displayed for unauthenticated users. Popular APIs are displayed on the Home page.

Users must sign on to the Developer Portal to see any APIs that are published with a visibility of **authenticated user** or **custom**.



## Create an account on the Developer Portal (1 of 2)

- If self-service is enabled in the API Manager portal configuration, a new account can be created from the menu in the Developer Portal
  - Select **Create an account** from the Developer Portal menu



Use an API through the Developer Portal

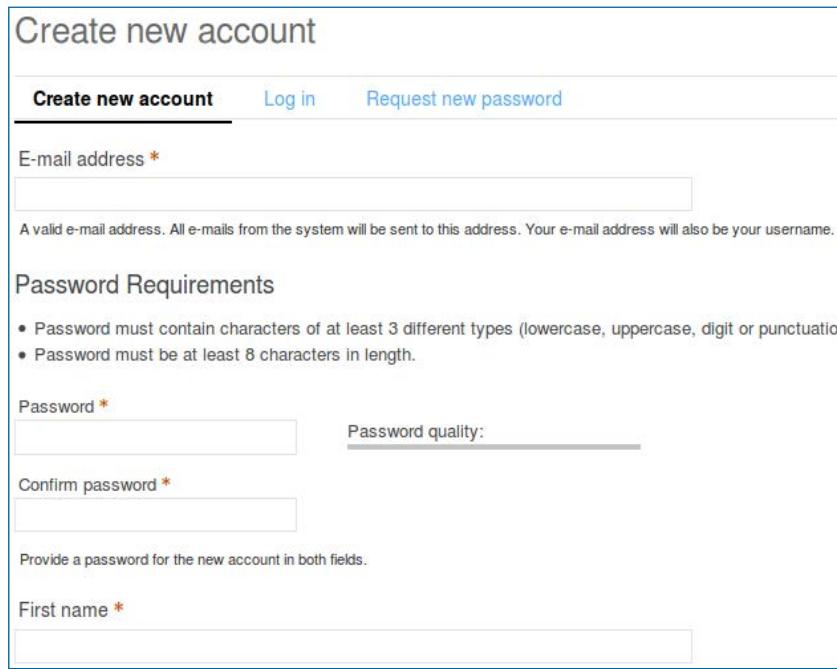
© Copyright IBM Corporation 2016

Figure 9-8. Create an account on the Developer Portal (1 of 2)

If self-service onboarding is enabled in the API Manager portal configuration, the Developer Portal displays an icon named Create an account. Developers or users can click this icon to create their own account on the Developer Portal.

## Create an account on the Developer Portal (2 of 2)

- Type in all required fields



**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 \*

Confirm password \*

Provide a password for the new account in both fields.

First name \*

Last name \*

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-9. Create an account on the Developer Portal (2 of 2)

A valid email address is a required field when creating a new account on the Developer Portal.

The email address becomes the user name that is used when signing on to the portal.

The user becomes the owner of a developer organization that they type when creating the new account.

You can only create your own developer organization and cannot join an existing developer organization with the create an account option.

Once you become the owner of the developer organization, you can add other users to the developer organization.



## Sign in to the Developer Portal

- Existing Portal users can sign on with their account
- The Portal administrator or Developer organization owners can add users to a Developer organization

The screenshot shows the 'User login' page of the IBM API Connect /dev portal. At the top, there are links for 'Create an account' and 'Login'. Below that is a navigation bar with 'Home', 'Getting Started', 'API Products', 'Blogs', 'Forums', and 'Support'. The main area is titled 'User login' and contains three input fields: 'Username \*' with the value 'student@think.com', 'Password \*' with several dots as the password, and a placeholder 'Enter your 192.168.225.20 username.' Below these fields is a 'Log in' button.

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-10. Sign in to the Developer Portal*

New developer organizations can be added from the Developers tab for the catalog in API Manager.

The API provider owner or administrator can add an existing portal user to a developer organization.



## Adding a user to a Developer Organization

- The organization owner can invite users to join the Developer organization

The screenshot shows the IBM API Connect /dev portal interface. On the left, a sidebar menu is open, showing options like 'My bookmarks', 'My organization' (which is highlighted with a red box and circled with a yellow circle labeled '1'), 'Create organization', and 'Log out'. The main content area is titled 'Kevin PortalUser' and has tabs for 'Manage' (which is underlined) and 'Analytics'. Under 'Manage', there's a section for 'Edit Organization' with fields for 'New user's e-mail address \*' (with a red box around it and circled with a yellow circle labeled '2') and 'Role' (with 'App Developer' selected). A button 'Invite new user' is at the bottom. In the top right corner of the main content area, there's a link 'Add a user' with a small user icon, also highlighted with a red box.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-11. Adding a user to a Developer Organization*

A developer organization owner can invite other users to become members of the developer organization.

When signed on, the organization owner selects **My organization** from the drop-down list of options alongside the user name on the Developer Portal.

The owner then selects **Add a user** and includes the new user's email address and the developer role.



## More APIs are visible to authenticated users

- API products that are visible to authenticated users are displayed when the user signs on to the Developer Portal
  - Select **API Products** from the menu to see the list of Products
- An extra menu item is displayed for authenticated users
  - Apps

 A screenshot of the IBM API Connect developer portal interface. The top navigation bar includes links for Home, Getting Started, API Products, Apps, Blogs, Forums, and Support, along with a search bar. The main content area displays two API products:
 

- think (v1.0.0)**: Represented by a blue circular icon with three stacked layers. Below it is the text "The think product" and a link to "APIs". To the right is a five-star rating icon followed by the text "No votes yet".
- hello-world (v1.0.0)**: Represented by a green circular icon with three stacked layers. Below it is the text "The hello-world product" and a link to "APIs". To the right is a five-star rating icon followed by the text "No votes yet".

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-12. More APIs are visible to authenticated users

Select the API Products tab to see all the available Products and APIs.

In the example, the think product and APIs are visible since the user has authenticated by signing on to the Developer Portal. The hello-world product and APIs that are published with public visibility are also visible.

Notice that the Apps tab is now visible on the menu. The Apps tab does not display on the Developer Portal unless the user is signed on.

## 9.2. Create an application that uses APIs

## Create an application that uses APIs

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-13. Create an application that uses APIs*

## Topics

- Browse and sign on to the Developer Portal to explore APIs
- Create an application that uses APIs
- Test an API on the Developer Portal
- Customize the Developer Portal

Use an API through the Developer Portal

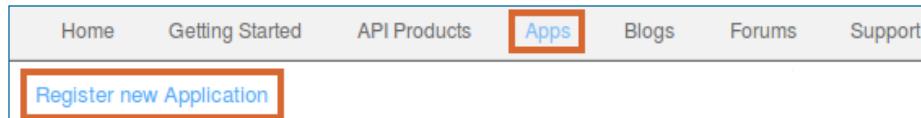
© Copyright IBM Corporation 2016

Figure 9-14. Topics

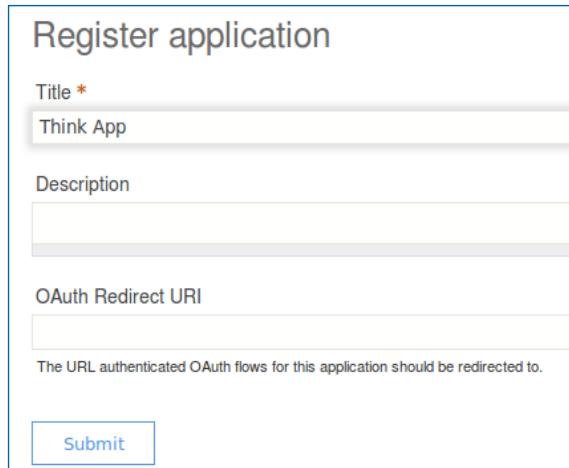


## Register an application (1 of 2)

1. To create an application that uses an API, select the **Apps** tab
  - Then, click **Register new Application**



2. Complete the Register application dialog. Then, click **Submit**



The image shows a "Register application" dialog box. It contains the following fields:

- Title \***: A text input field containing "Think App".
- Description**: A text area field.
- OAuth Redirect URI**: A text input field.
- The URL authenticated OAuth flows for this application should be redirected to.
- Submit**: A blue button at the bottom of the form.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-15. Register an application (1 of 2)



## Register an application (2 of 2)

- Click the check boxes to display the client secret and client ID when they are to be used by the application

A screenshot of the IBM Cloud developer portal. At the top, there's a success message: "Application created successfully." and "Your client secret is: rY1sL6qE5pP3aO2tL0jD6nH6oP1cM6hK6iW6eJ1rQ1eD3pD3kA". A checkbox labeled "Show Client Secret" is checked. Below this, there's a summary of the application "Think App": it has a brown circular icon with a smartphone and gear, was created on "2016-05-24", and has a "Description" section. Under "Client Credentials", the "Client ID" is listed as "785a420e-011b-41be-97f4-e4f356829d5d". A "Show" checkbox is checked next to the Client ID, and a "Reset" button is available. Navigation links "[Back to Apps](#)" and "[Update](#)" are also present.

< Back to Apps

Think App

Update

2016-05-24

Description

Client Credentials

Client ID  
785a420e-011b-41be-97f4-e4f356829d5d  Show

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-16. Register an application (2 of 2)

By default the application does not display the client secret. Click Show Client Secret and record this value if the application uses the client secret and client ID during authentication.



## Subscribe the application to a plan (1 of 6)

- From the API Products tab, select a Product
- Go to **Plans**

A screenshot of the IBM API Connect /dev interface. The top navigation bar includes "Home", "Getting Started", and "API Products". Below this, there's a sidebar with "think 1.0.0" and a list of APIs: "APIs", "inventory 1.0.0", "financing 1.0.0", "logistics 1.0.0", and "oauth 1.0.0". On the right, there's a main panel for "think 1.0.0" showing a blue circular icon with three white layers, a rating of "No votes yet", and a description "Description The think product". At the bottom of the sidebar, the word "Plans" is highlighted with a red box.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-17. Subscribe the application to a plan (1 of 6)



## Subscribe the application to a plan (2 of 6)

- In the list of available plans, select a plan. Then, click **Subscribe**

Plans		
	Gold	Silver
financing 1.0.0		
logistics 1.0.0	unlimited	100 per hour
oauth 1.0.0		
financing 1.0.0	unlimited	100 per hour
logistics 1.0.0	unlimited	100 per hour
oauth 1.0.0	unlimited	100 per hour
	<a href="#">Subscribe</a>	<a href="#">Subscribe</a>

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

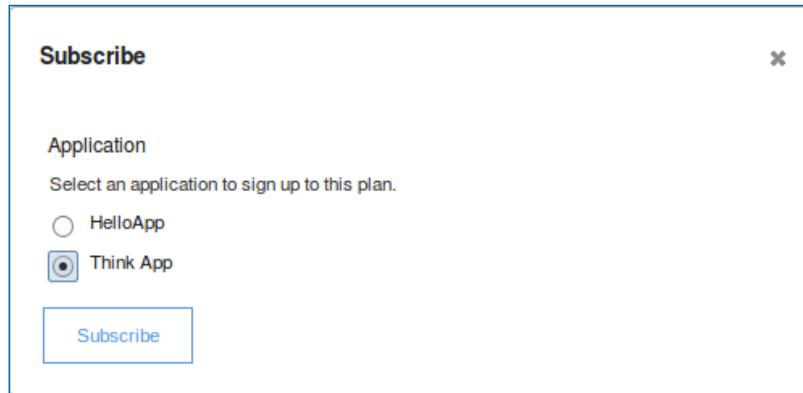
Figure 9-18. Subscribe the application to a plan (2 of 6)

An application plan subscription allows the application to invoke API resources that are exposed by the plan.

A plan is collection of REST API operations and SOAP API WSDL operations. In the Developer Portal you can browse and select the most appropriate plan to use with your application.

## Subscribe the application to a plan (3 of 6)

- Select the application that you want to subscribe to the plan
- Click **Subscribe**



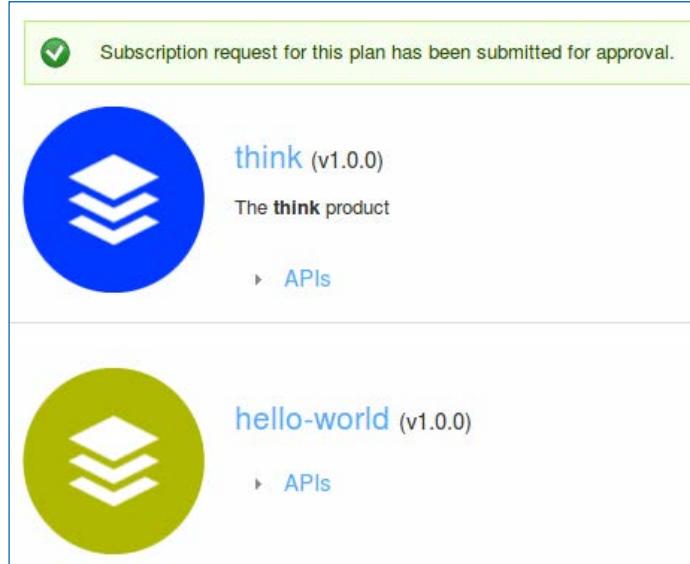
Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-19. Subscribe the application to a plan (3 of 6)

## Subscribe the application to a plan (4 of 6)

- A message is displayed indicating that the subscription request has been submitted for approval



Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-20. Subscribe the application to a plan (4 of 6)



## Subscribe the application to a plan (5 of 6)

- The application is now subscribed to the plan

A screenshot of the IBM Cloud developer portal. The page shows a summary of an application named "Think App". It includes a circular icon with a smartphone and gear, a creation date of "2016-05-24", and a "Description" field. Below this is a "Client Credentials" section with fields for "Client ID" (containing a redacted string) and "Client Secret" (also redacted), each with "Show", "Verify", and "Reset" buttons. To the right of these fields is a small edit icon. At the bottom of the page, under the heading "Subscriptions", there is a single entry: "think (v1.0.0) silver" with an "Unsubscribe" button to its right. The entire interface has a clean, modern design with a light blue header and white background.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-21. Subscribe the application to a plan (5 of 6)



## Subscribe the application to a plan (6 of 6)

- Expand the plan to see the list of subscribed API operations

The screenshot shows the 'Subscriptions' section of the IBM Developer Portal. At the top, it lists a subscription for 'think (v1.0.0) silver' with an 'Unsubscribe' link. Below this, the 'logistics (v1.0.0)' plan is expanded, revealing two API operations:

Verb	Path	Description	Rate Limit
GET	/shipping		100 per hour
GET	/stores		100 per hour

At the bottom of the expanded plan, there is another link labeled 'oauth (v1.0.0)'.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-22. Subscribe the application to a plan (6 of 6)

The subscriptions for an application are listed below the application. The plan name and the APIs that are contained by the plan are displayed.

Expanding one of the APIs displays the API operations and rate limits.

## 9.3. Test an API on the Developer Portal

## Test an API on the Developer Portal

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-23. Test an API on the Developer Portal*

## Topics

- Browse and sign on to the Developer Portal to explore APIs
- Create an application that uses APIs
-  Test an API on the Developer Portal
- Customize the Developer Portal

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-24. Topics*



## APIs that can be tested on the Developer Portal

- Only APIs that have Testable enabled at design time can be tested on the Developer Portal

The screenshot shows the 'Design' tab selected in the top navigation bar. On the left, there's a sidebar with options like Info, Host, Base Path, Schemes, Consumes, Produces, Lifecycle (which is currently selected), Policy Assembly, Security Definitions, and Security. The main content area is titled 'Lifecycle' and contains a 'Phase' section with three toggle switches: 'Enforced' (Enforce API using a gateway), 'Testable' (Allow the API to be tested using the developer portal test tool, which is highlighted with a yellow box), and 'CORS' (Enable CORS access control). A callout box points to the 'Testable' option with the text 'The Lifecycle option in **API Manager**'.

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-25. APIs that can be tested on the Developer Portal

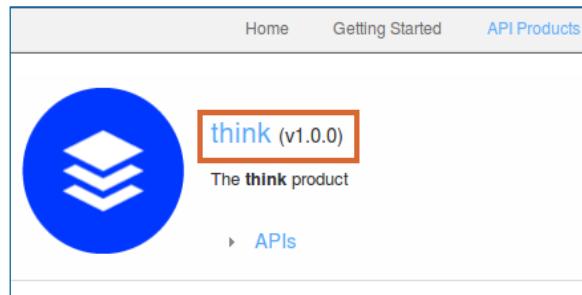
The image on the page is taken from the Lifecycle part for an API in the API Designer.

The enforced, testable, and CORS access must all be enabled to allow developers to test the API in the Developer Portal.



## Test the API on the Developer Portal (1 of 4)

- In the API Products view, click the link for the Product whose APIs you want to test



Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-26. Test the API on the Developer Portal (1 of 4)

To test APIs on the Developer Portal, select the API Products tab in the Developer Portal.

Then, click the link for the Product whose APIs you want to test.



## Test the API on the Developer Portal (2 of 4)

- Select the API from the list of APIs
- Click the API operation that you want to test

The screenshot shows the IBM API developer portal interface. On the left, there's a sidebar with categories like "think 1.0.0", "APIs", "inventory 1.0.0", "financing 1.0.0", and "logistics 1.0.0". The "APIs" category is selected. In the main content area, under "Paths", the "/shipping" endpoint is selected. Below it, the "GET /shipping" operation is highlighted. To the right, there's a "Example Request" section with tabs for "cURL", "Ruby", "Python", "PHP", "Java", "Node", "Go", and "Swift". The "cURL" tab is selected, displaying the following command:

```
curl --request GET \
--url 'https://192.168.225.52/sales/sb/
logistics/shipping?zip=REPLACE_THIS_VALUE
' \
--header 'accept: application/json' \
--header 'content-type: application/json' \
--header 'x-ibm-client-id: REPLACE_THIS_KEY'
```

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-27. Test the API on the Developer Portal (2 of 4)*

The list of APIs for the Product is displayed.

Select the API and operation that you want to test.

The example test request is displayed.

## Test the API on the Developer Portal (3 of 4)

- Scroll to Try this operation
- Select the appropriate application name in the Client ID field
- Type any required parameters
- Click **Call operation**

Try this operation

`https://192.168.225.52/sales/sb/logistics/shipping`

Identification

Client ID	Think App
-----------	-----------

Headers

content-type	application/json
accept	application/json

Parameters

zip	15209
-----	-------

**Call operation**

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-28. Test the API on the Developer Portal (3 of 4)

Scroll to the Try this operation area in the browser.

Select the appropriate application from the Client ID drop-down list.

Type any required parameters.

Click Call operation.



### Note

The Developer Portal matches the Client ID with the client ID that is generated by the application.

Ensure that you select the appropriate application from the list of applications in the Client ID drop down, otherwise you get a message telling you that the Client ID is invalid.

## Test the API on the Developer Portal (4 of 4)

- The result is displayed

Response

```
200 OK
Content-Type: application/json
X-Global-Transaction-ID: 193184
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 98
{
  "xyz": {
    "next_day": "22.03",
    "two_day": "15.07",
    "ground": "11.60"
  },
  "cek": {
    "next_day": "20.03",
    "two_day": "13.70",
    "ground": "10.54"
  }
}
```

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-29. Test the API on the Developer Portal (4 of 4)

The result of calling the operation is displayed in the browser.

If an error is displayed, check your parameters and ensure that the appropriate back-end services are running.

## 9.4. Customize the Developer Portal

## Customize the Developer Portal

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-30. Customize the Developer Portal*

## User story and role when customizing the Developer Portal

- User story: As a Portal Administrator, you want to customize the user interface of the Developer Portal
  - Role: Portal Administrator



[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-31. User story and role when customizing the Developer Portal*

## Topics

- Browse and sign on to the Developer Portal to explore APIs
- Create an application that uses APIs
- Test an API on the Developer Portal



Customize the Developer Portal

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-32. Topics



## Customize the Developer Portal

- Change the theme for the portal
  - Import a pre-built theme that is packaged as an archive file
  - Contains stylesheets, image files, templates, and icons
- Create a theme by customizing an existing theme, such as the IBM API Connect theme
- You must be signed on to the Developer Portal with an administrator role to install and configure a custom theme

The screenshot shows the login interface for the IBM API Connect /dev portal. At the top, there's a header bar with the IBM API Connect logo and the URL "IBM API Connect /dev". To the right of the URL are links for "Create an account" and a highlighted "Login" button. Below the header is a "User login" section. It includes links for "Create new account", "Log in" (which is underlined), and "Request new password". The "Username \*" field contains "admin", with a placeholder "Enter your 192.168.225.20 username." below it. The "Password \*" field contains a masked password, with a placeholder "Enter the password that accompanies your username." below it. At the bottom of the form is a "Log in" button.

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-33. Customize the Developer Portal*

In the lab exercise, you install and configure a custom theme by loading the theme into the Developer Portal.

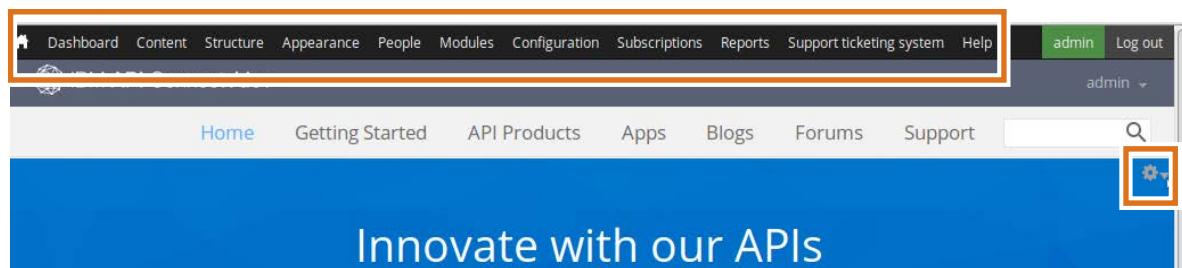
The custom theme is a pre-built theme that contains style sheets, image files, templates, and icons.

A new theme can be created by taking an existing theme, such as the default theme that comes with the Developer Portal and customizing the style elements, image files, templates, and icons.

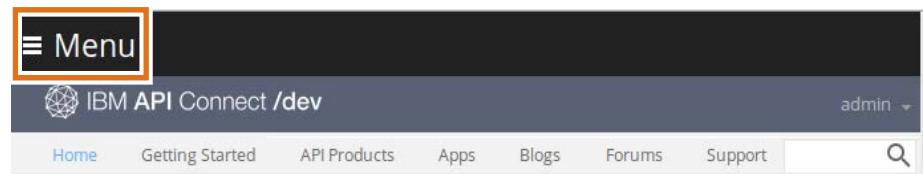


## The administrative menu in the Developer Portal

- The administration menu is displayed when the admin user signs on
  - More pop-up menu icons are also displayed



- The administration menu is responsive and is collapsed to a menu icon when the page is resized smaller



[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-34. The administrative menu in the Developer Portal

When you sign on to the Developer Portal with an administrator role, a menu with administrator actions is displayed at the top of the page.



## Install a custom theme (1 of 5)

- Select **Appearance** from the administration menu
  - Then, select **Install new theme**

The screenshot shows the Drupal Administration interface. The top navigation bar includes links for Dashboard, Content, Structure, Appearance (which is highlighted in blue), People, Modules, and Configuration. A dropdown menu is open under the Appearance link, showing options: Install new theme (which is highlighted with a blue background), Update, Settings, and Uninstall. Below this, the main content area is titled 'Appearance' and shows the path 'Home » Administration » Appearance'. It contains instructions for finding modules and themes on drupal.org and installing from a URL or file archive. A 'Browse...' button is shown with 'No file selected.' and a note about selecting a tar.gz file. A large green 'Install' button is at the bottom.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

*Figure 9-35. Install a custom theme (1 of 5)*

To install a theme, select **Appearance** from the administration menu. Then, select **Install new theme**.

Browse to the location of the theme archive file and open it.

Then, click **Install**.

## Install a custom theme (2 of 5)

- The theme is installed
- Click the Enable newly added themes link

 Installation was completed successfully.

**think\_ibm\_connect\_theme**

- Installed *think\_ibm\_connect\_theme* successfully

**Next steps**

- [Enable newly added themes](#)
- [Administration pages](#)

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-36. Install a custom theme (2 of 5)

When the theme is installed, click **Enable newly added themes**.

## Install a custom theme (3 of 5)

- Scroll down in the Appearance menu to the newly added theme
- Click the **Enable and set default** link



Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-37. Install a custom theme (3 of 5)

Scroll through the list of available themes until you find the newly-imported theme. Then, click **Enable and set default**.



## Install a custom theme (4 of 5)

- Click the **Settings** tab

The screenshot shows the "Appearance" settings page in a Drupal-style interface. The "Settings" tab is highlighted with an orange border. A success message states "thinkibm\_connect is now the default theme." A warning message says "No update information available. Run cron or check manually." Below these messages, there's a note about setting and configuring the default theme. A blue button labeled "Install new theme" is visible. The "ENABLED THEMES" section displays the "thinkibm\_connect 7.43 (default theme)" with a preview image and a brief description: "A sample theme designed for IBM API Connect but otherwise it is a minimal, Drupal 7 that uses HTML5 and Adaptivetheme 7.x-3.x base theme." A "Settings" link is also present.

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-38. Install a custom theme (4 of 5)

The imported theme is now the default theme.

Next, you save the configuration from the Settings tab of the Appearance menu.



## Install a custom theme (5 of 5)

- Select the options in toggle display
- Click the **Save configuration**
- The default theme is saved

The screenshot shows a web-based administrative interface for managing site settings. At the top, there are tabs: List, Update, Settings (which is selected), and Uninstall. Below the tabs, there are links: Global settings, Adminimal, IBM API Connect, and thinkibm\_connect. The main content area is titled "Global settings". It contains sections for "Toggle display", "Logo image settings", and "Shortcut icon settings". Under "Toggle display", there are checkboxes for Logo, Site name, Site slogan, Shortcut icon, Main menu, and Secondary menu, all of which are checked. Under "Logo image settings", there is a checkbox for "Use the default logo" which is checked. Under "Shortcut icon settings", there is a checkbox for "Use the default shortcut icon" which is also checked. At the bottom of the page is a green "Save configuration" button.

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-39. Install a custom theme (5 of 5)

Click **Save configuration** from the Settings tab of the Appearance menu.

When done, click the Home icon on the administrative menu to display the new theme.



## Customize the Welcome Banner (1 of 3)

- Select **Content** from the administration menu
- Then, select **Blocks**
- **Edit** the Welcome Banner

A screenshot of the IBM Content Management interface. The top navigation bar includes links for Home, Getting Started, API Products, Apps, Content (which is highlighted in grey), and Blocks. Below the navigation, a breadcrumb trail shows "Home » Administration » Content". A prominent "Add Block" button is visible. On the left, there's a sidebar titled "Filters" with a dropdown menu showing "Banner", "Popular API", "Popular Product", and "Social Block". The main area displays a table of blocks. The columns are "Title", "Type", and "Operations". One row is selected, showing "Welcome Banner" in the Title column and "banner\_block" in the Type column. The "Operations" column contains two buttons: "Edit" and "Delete", with the "Edit" button highlighted by a red box. At the bottom of the table, there are buttons for "Items Per Page" (set to 50), "Filter", and "Reset".

Title	Type	Operations
Welcome Banner	banner_block	<a href="#">Edit</a> <a href="#">Delete</a>

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-40. Customize the Welcome Banner (1 of 3)

Customize the Welcome Banner from the **Content** option of the administration menu. Then, select **Blocks**.

Next, click **Edit** to edit the Welcome Banner



## Customize the Welcome Banner (2 of 3)

- Browse to the location of the welcome image file
- Upload the file

**Edit Block**

<b>View</b>	<b>Edit Block</b>	<b>Revisions</b>	<b>Delete Block</b>
<b>Label *</b>			
Welcome Banner			
Name that displays in the admin interface			
<b>Title</b>			
<input type="text"/> The Title of the block. The <em>, <strong>, <i> and <b> HTML tags are allowed,         </b></i></strong></em>			
<b>Content</b>			
			
<b>Path:</b> h1 » span <b>Disable rich-text</b>			
<b>Text format</b> <b>Full HTML</b> <ul style="list-style-type: none"> <li>• Web page addresses and e-mail addresses turn into links automatically.</li> <li>• Lines and paragraphs break automatically.</li> </ul>			
<b>Image</b>			
<input type="button" value="Browse..."/> thinkibm_welcome_banner.png <input type="button" value="Upload"/>			

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-41. Customize the Welcome Banner (2 of 3)

In the Edit Block dialog, browse to the location of the welcome image file.

Then, click **Upload**.



## Customize the Welcome Banner (3 of 3)

- Save the changes

The screenshot shows a configuration interface for a welcome banner. At the top, there's a preview thumbnail of the image, its file name ("thinkibm\_welcome\_banner.png"), its size ("205.1 KB"), and a "Remove" button. Below this is a section for "Background Image". Under "View Mode \*", a dropdown menu is set to "Default". There are two checked checkboxes: "Create new revision" and "Set Revision as default". A "Log message" input field is present, with a placeholder text about explaining changes. At the bottom are "Save" and "Delete" buttons.

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

Figure 9-42. Customize the Welcome Banner (3 of 3)

Save the changes.

IBM Training

IBM

## Developer Portal with the new Welcome Banner and theme

The screenshot shows the IBM Developer Portal interface. At the top, there's a blue header bar with the 'IBM Training' logo on the left and the 'IBM' logo on the right. Below the header is a navigation bar with links: Dashboard, Content, Structure, Appearance (which is highlighted in blue), People, Modules, Configuration, Subscriptions, Reports, Support ticketing system, Help, admin (in a green box), and Log out. A dropdown menu for 'admin' is open. The main content area features a dark banner with the text 'Innovate with our APIs' and a subtext 'Welcome to our API portal where you will find a great selection of APIs for your awesome innovative apps'. Below the banner is a globe icon with glowing blue lines representing network connections. The page also includes sections for 'Popular APIs' and 'Latest forum posts and tweets'.

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-43. Developer Portal with the new Welcome Banner and theme

## Unit summary

- Describe how to access the Developer Portal
- Review the differences between the Developer Portal with and without signing on
- Describe how users are added to the Developer Portal
- Describe how to create an application on the Developer Portal
- Illustrate how applications can be subscribed to plans
- Explain how applications are tested on the Developer Portal
- Describe how to add a custom theme to the Developer Portal

[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-44. Unit summary*

## Review questions

1. True or False: Public APIs are displayed on the public interface and the interface for signed on users of the Developer Portal
  
2. Where is self-service onboarding set for the Developer Portal?
  - A. On the Lifecycle option in API Designer
  - B. On the My Organization option in the Developer Portal
  - C. On the Developer Organization tab in API Manager
  - D. In the Portal Configuration area in API Manager



[Use an API through the Developer Portal](#)

© Copyright IBM Corporation 2016

*Figure 9-45. Review questions*

Write your answers here:

- 1.
- 2.
- 3.

## Review answers

1. True or False: Public APIs are displayed on the public interface and the interface for signed on users of the Developer Portal

The answer is True.

Public APIs are displayed for anonymous users and authenticated users of the Developer Portal



2. Where is self-service onboarding set for the Developer Portal?

- A. On the Lifecycle option in API Designer
- B. On the My Organization option in the Developer Portal
- C. On the Developer Organization tab in API Manager
- D. In the Portal Configuration area in API Manager

The answer is D.

## Exercise: Consumer Experience

### Lab 7

Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-47. Exercise: Consumer Experience

## Exercise objectives

- Add a custom theme to change the overall layout of the developer portal
- Register an application with the developer portal
- Subscribe to a plan to use an API
- Test an API from the developer portal
- Use an API from a sample test application



Use an API through the Developer Portal

© Copyright IBM Corporation 2016

Figure 9-48. Exercise objectives

---

# Unit 10. Exercise case study

## Estimated time

00:30

## Overview

This unit describes the scenario that is used throughout this course as a basis for exercises.

## Unit objectives

- Describe the architecture of the application
- Define how the human task components fit into the scenario
- Explain how each lab exercise contributes to the development of a working application

Exercise case study

© Copyright IBM Corporation 2016

*Figure 10-1. Unit objectives*

## Lab 1: Hello World

- Use the apic command line
- Create a hello-world loopback application
  - apic loopback hello-world
  - Generates a hello-world Node application
- Review the generated Node modules, folders, and files
- Open the API Designer
  - apic edit
- Launch the hello-world application
- Generate sample data
- Test the hello-world application
  
- The hello-world application is *not used* in the later labs
  - Not a component of the use case scenario that is developed in the later labs

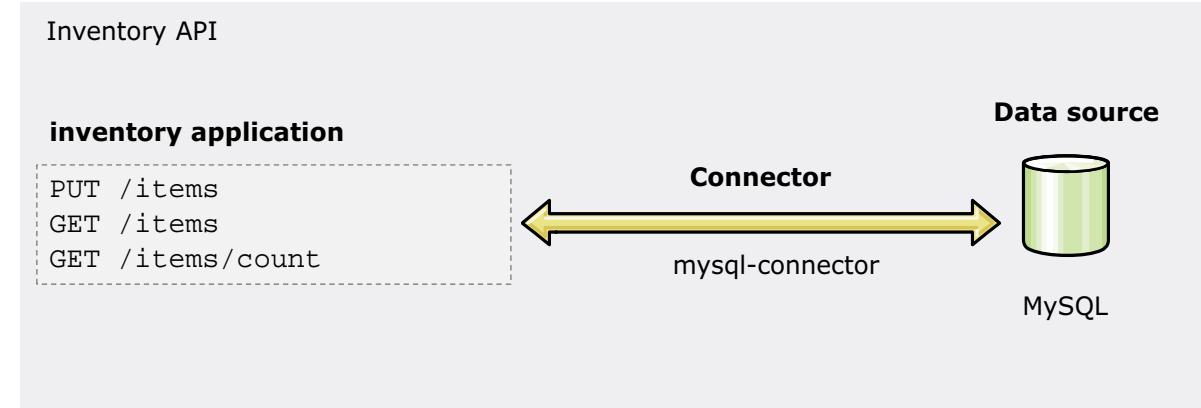
[Exercise case study](#)

© Copyright IBM Corporation 2016

*Figure 10-2. Lab 1: Hello World*

## Lab 2: Create the Inventory API (1 of 2)

- Generate an API named **Inventory** with LoopBack
  1. Create a loopback data source and connector for the MySQL database
  2. Create a loopback model named **item**
    - Reference the mysql-connector from the **item** model
    - Create the properties: name, description, img, img\_alt, price, rating
  3. Start the inventory loopback application
  4. Explore the GET /items and GET /items/count operations
    - Returns 12 items from the MySQL database



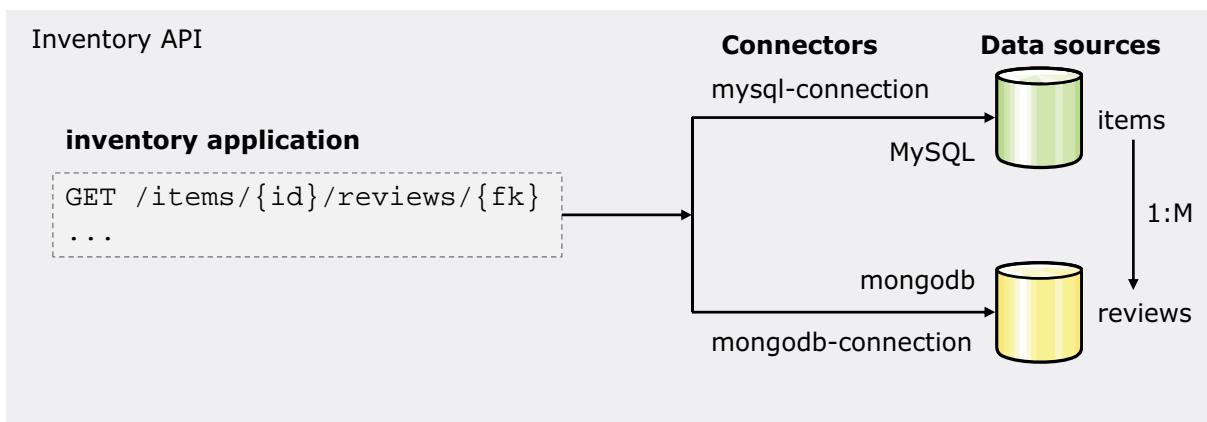
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-3. Lab 2: Create the Inventory API (1 of 2)

## Lab 2: Create the Inventory API (2 of 2)

- Add another model to the **Inventory API**
  1. Create a loopback data source and connector for the mongodb database
  2. Create a loopback model named **review**
    - Reference the mongodb-connection from the **review** model
    - Create the properties: date, reviewer-name, reviewer-email, comment, rating
  3. Create a 1:M relationship between item and review
  4. Explore the Paths in the Inventory API
  5. Validate the lab source code



Exercise case study

© Copyright IBM Corporation 2016

Figure 10-4. Lab 2: Create the Inventory API (2 of 2)

Before moving on to the next lab, a simple script is provided for you that validates your source files against those of a completed lab.

From the terminal, type validate\_lab 2 at the end of the lab.

## Lab 3: Customize and publish the Inventory API (1 of 2)

- Add post-processing JavaScript to the items in the **Inventory API**
  1. Edit **items.js**
  2. Include new remote hook function that runs after a new review is submitted for an item
    - Calculate the average of all reviews for the item
    - Update the item rating in the MySQL data source

```
module.exports = function (Item) {
  Item.afterRemote('prototype.__create__reviews', function (ctx,
remoteMethodOutput, next
  var itemId = remoteMethodOutput.itemId;
  console.log("calculating new rating for item: " + 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 = Math.round((ratingSum / reviewCount) * 100) / 100;
  ...
}
```

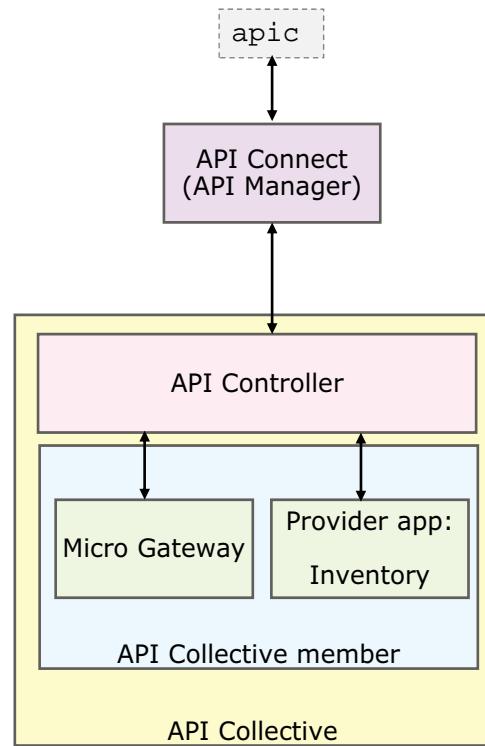
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-5. Lab 3: Customize and publish the Inventory API (1 of 2)

## Lab 3: Customize and publish the Inventory API (2 of 2)

- Publish the **Inventory** API/app to a Collective
  1. In API Manager, add the Inventory app to the API Collective
  2. Configure the Developer Toolkit to communicate with API Connect (APIM)
    - In API Manager, copy the hyperlink for the Inventory app
    - In the CLI inventory folder, set the app variable to the hyperlink value
      - apic config:set app=apic-app://\$(host)/orgs/\$(org)/apps/\$(app)
  3. Start the API Collective server
    - Type: wlpn-controller start
  4. Login to API Manager
    - apic login
  5. Publish the inventory application
    - apic apps:publish
    - Copy the generated header app-id for later



Exercise case study

© Copyright IBM Corporation 2016

Figure 10-6. Lab 3: Customize and publish the Inventory API (2 of 2)

Before moving on to the next lab, a simple script is provided for you that validates your source files against those of a completed lab.

From the terminal, type validate\_lab 3 at the end of the lab.

## Lab 4: Secure the Inventory API (1 of 3)

- Add a new **oauth** provider API
  1. Create an **oauth** API for obtaining access tokens
  2. Configure the Oauth 2 settings as:
    - Collect credentials using: basic
    - Client type: Confidential
    - Scope: Inventory
    - Description: Access to the Inventory API
    - Authenticate application users with: Authentication URL
    - Authentication URL: `https://<host>:<port>/<path>`

### **oauth API operations**

```
GET /authorize  
POST /authorize  
POST /tokens
```

Figure 10-7. Lab 4: Secure the Inventory API (1 of 3)

## Lab 4: Secure the Inventory API (2 of 3)

- Add the security definitions to the **Inventory API**
  1. Edit the **Inventory API**
    - Select **oauth** for the security definition
  2. Add properties for catalog-specific backend URLs
  3. Add an **app-id** property to the Sandbox catalog
    - Paste the value from the host header app-id that was generated when publishing the app in an earlier lab

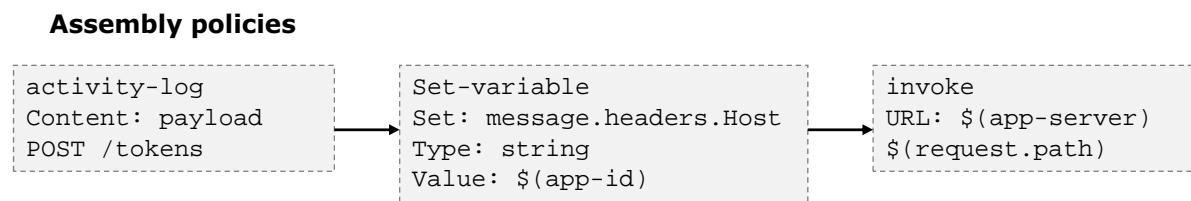
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-8. Lab 4: Secure the Inventory API (2 of 3)

## Lab 4: Secure the Inventory API (3 of 3)

- Define the API processing behavior
  1. Create an assembly for the Inventory API
  2. Add DataPower Gateway policies
    - activity-log
    - set-variable
    - invoke



[Exercise case study](#)

© Copyright IBM Corporation 2016

Figure 10-9. Lab 4: Secure the Inventory API (3 of 3)

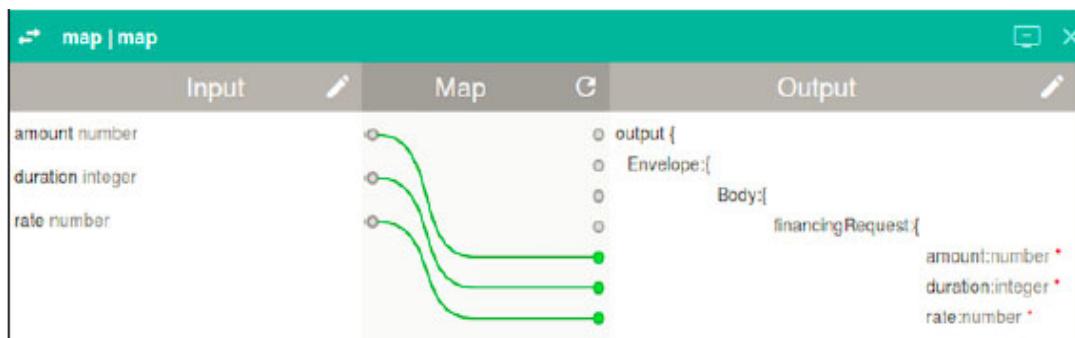
Before moving on to the next lab, a simple script is provided for you that validates your source files against those of a completed lab.

From the terminal, type validate\_lab 4 at the end of the lab.

## Lab 5: Advanced API assembly (1 of 2)

- Define a new API named financing
  1. Create an assembly for the financing API that converts a SOAP service to a REST service
  2. Add DataPower Gateway policies
    - activity-log
    - map
    - invoke
    - gateway script
    - xml-to-json

### Map policy



Exercise case study

© Copyright IBM Corporation 2016

Figure 10-10. Lab 5: Advanced API assembly (1 of 2)

## Lab 5: Advanced API assembly (2 of 2)

- Define a new API named logistics
  - 1. Add properties rates, shipping, store\_location
  - 2. Add paths:
    - /shipping
    - /stores
  - 3. Create an assembly for the logistics API that uses an existing REST service
  - 4. Add DataPower Gateway policies
    - activity-log
    - operation-switch
    - invoke
    - map
    - invoke geolocate
    - gateway script

[Exercise case study](#)

© Copyright IBM Corporation 2016

Figure 10-11. Lab 5: Advanced API assembly (2 of 2)

Before moving on to the next lab, a simple script is provided for you that validates your source files against those of a completed lab.

From the terminal, type validate\_lab 5 at the end of the lab.

## Lab 6: Publishing an API Product

- Configure the Product
  1. Edit the default Product that was created during the creation of the inventory loopback application
  2. Change the Product properties and visibility
    - Name the Product think
  3. Add APIs to the Product
    - financing
    - logistics
    - oauth
- Configure the Plans
  1. Edit the default Plan that was created during the creation of the inventory loopback application
  2. Change the Plan properties
    - Name the Plan silver
    - Rate limit: 100 /hour
  3. Create a new Plan named gold with unlimited access to APIs
- Publish the Product
  - Target Catalog: Sandbox

[Exercise case study](#)

© Copyright IBM Corporation 2016

Figure 10-12. Lab 6: Publishing an API Product

Before moving on to the next lab, a simple script is provided for you that validates your source files against those of a completed lab.

From the terminal, type validate\_lab 6 at the end of the lab.

## Lab 7: Customize the Developer Portal

- Customize the Developer Portal
  - Sign on with user admin
  - Change the custom theme
- Register an application as a developer
  - Sign on to the portal as a user in the application developer role
  - Register an application
  - Subscribe to a plan
- Test the Product APIs from the Developer Portal
  - Try the operations for APIs in the Product
- Test the Product APIs from a consumer web application
  - Install the sample consumer web application
  - Test the APIs from the consumer web application

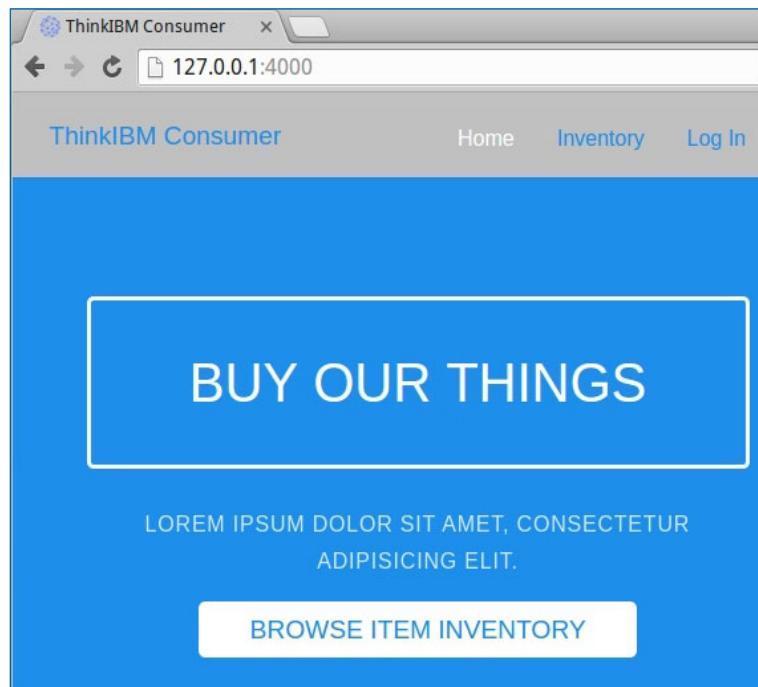
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-13. Lab 7: Customize the Developer Portal

# IBM Training

## Consumer Application (1 of 10)



[Exercise case study](#)

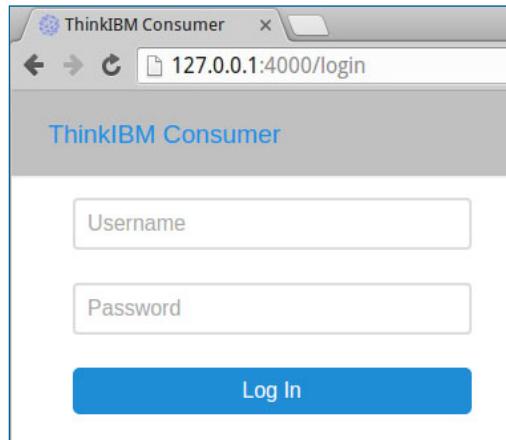
© Copyright IBM Corporation 2016

Figure 10-14. Consumer Application (1 of 10)

This is the home page for the consumer application that shows the inventory application that is built in the course exercises.

IBM Training 

## Consumer Application (2 of 10)



Exercise case study

© Copyright IBM Corporation 2016

Figure 10-15. Consumer Application (2 of 10)

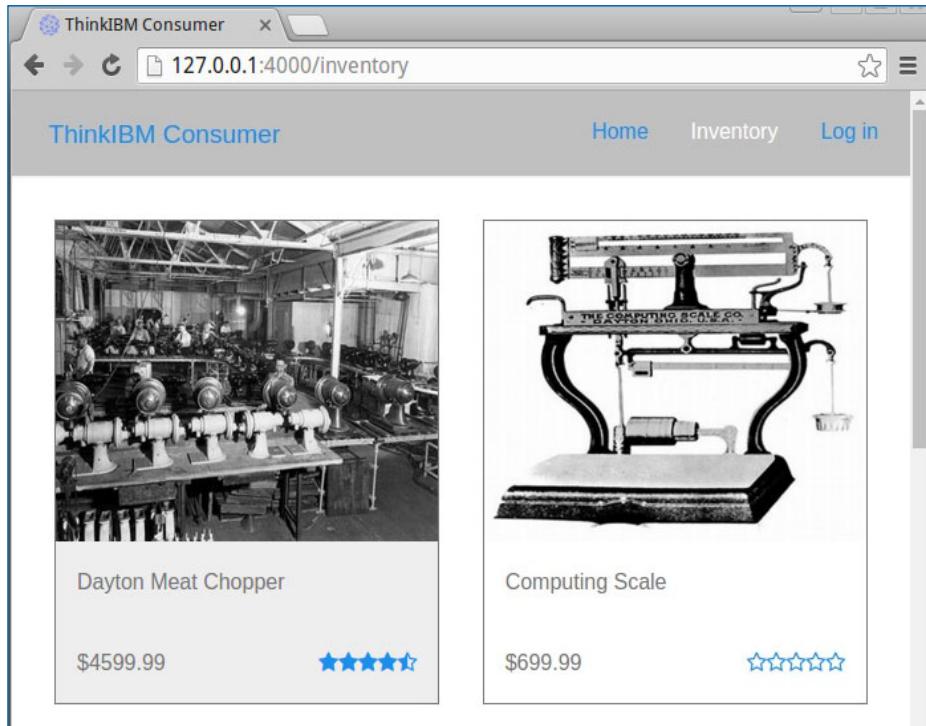
Sign on with any user ID and password.

The Oauth authenticates with an authentication URL.

If there is anything wrong with the application, it fails to authenticate.

# IBM Training

## Consumer Application (3 of 10)



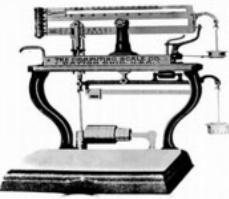
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-16. Consumer Application (3 of 10)

If you finish the labs, at the end of lab 7 you run the consumer application that authenticates the user (any user ID and password) and displays the inventory page, if the labs have been completed successfully.

## Consumer Application (4 of 10)



**Computing Scale**

5 stars 

\$699.99

[Calculate Monthly Payment](#)

---

**Product Description**

In 1885 Julius Pitrat of Gallipolis, Ohio, patented the first computing scale. In 1890 the Remington Typewriter Company of Dayton, Ohio, purchased Pitrat's patents and incorporated them into their business. And four years after that, The Computing Scale Company was formed. In 1911, the Computing Scale Company merged with the International Business Machines Corporation to form the Computing-Tabulating-Recording Company, a

---

**Customer Reviews (0)**

Exercise case study

© Copyright IBM Corporation 2016

Figure 10-17. Consumer Application (4 of 10)

Notice that the Computing Scale has no reviews.

IBM Training 

## Consumer Application (5 of 10)

Customer Reviews (0) [Write a review](#)

Rating

Jane

jdoe@example.com

semi-terrific

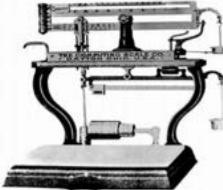
[Exercise case study](#)

© Copyright IBM Corporation 2016

Figure 10-18. Consumer Application (5 of 10)

In the Customer Reviews area, you can write a review and submit your rating.

## Consumer Application (6 of 10)



**Computing Scale**

★★★★★

\$699.99

[Calculate Monthly Payment](#)

---

**Product Description**

In 1885 Julius Pitrat of Gallipolis, Ohio, patented the first computing scale. In 1900, the National Scale Company of Dayton, Ohio, purchased Pitrat's patents and incorporated them into their product line. In 1911, the Computing Scale Company merged with the International Business Machines Corporation to form the Computing-Tabulating-Recording Company, a division of IBM.

---

**Customer Reviews (1)**

★★★★★ by Jane on Thu Jul 07 2016

"semi-terrific"

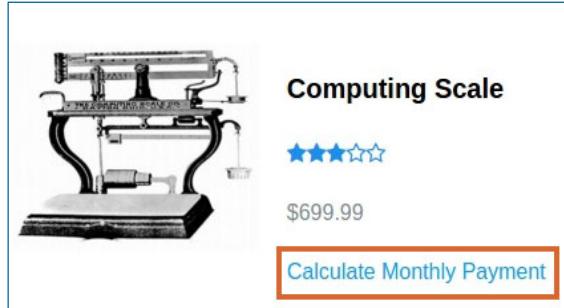
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-19. Consumer Application (6 of 10)

The review is added to the item and the average review rating is displayed.

## Consumer Application (7 of 10)



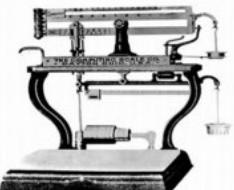
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-20. Consumer Application (7 of 10)

The user can click the Calculate Monthly Payment to calculate a payment.

## Consumer Application (8 of 10)



**Computing Scale**

★★★★★

\$699.99

Monthly Payment: \$30.37 at 3.9% for 24 months

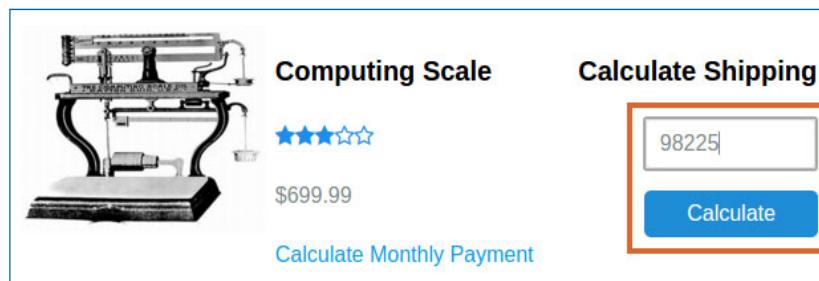
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-21. Consumer Application (8 of 10)

The payment is calculated and displayed for the item.

## Consumer Application (9 of 10)



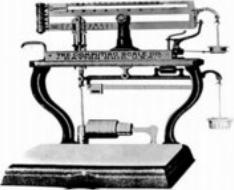
Exercise case study

© Copyright IBM Corporation 2016

Figure 10-22. Consumer Application (9 of 10)

Type in a zip code to calculate shipping costs.

## Consumer Application (10 of 10)



**Computing Scale**

★★★★★  
\$699.99  
[Calculate Monthly Payment](#)

**Calculate Shipping**

XYZ	Next Day	\$39.38
	Two Day	\$26.95
	Ground	\$20.73
<hr/>		
CEK	Next Day	\$35.80
	Two Day	\$24.50
	Ground	\$18.84

[Pickup from nearest store](#)

Exercise case study

© Copyright IBM Corporation 2016

Figure 10-23. Consumer Application (10 of 10)

The shipping cost is calculated and displayed for the item. Click Pickup from nearest store to display the map for the nearest store location.

## Unit summary

- Describe the architecture of the application
- Define how the human task components fit into the scenario
- Explain how each lab exercise contributes to the development of a working application

Exercise case study

© Copyright IBM Corporation 2016

*Figure 10-24. Unit summary*

---

# Unit 11. Course summary

## Estimated time

00:10

## 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 2016

*Figure 11-1. Unit objectives*

## Course objectives

- Explain the role of IBM API Connect V5.0 in a digital ecosystem
- Identify the API creation, run, manage, and secure features in IBM API Connect V5
- Create an API with the API Connect toolkit
- Define an API interface with the API Designer
- Call an existing system API
- Implement an API as a Node.js LoopBack application
- Secure an API with OAuth 2.0 authorization

[Course summary](#)

© Copyright IBM Corporation 2016

*Figure 11-2. Course objectives*

## Course objectives

- Assemble API message flows in API Designer
- Publish API and Products with API Management Server
- Create an application on the Developer Portal that uses an API

[Course summary](#)

© Copyright IBM Corporation 2016

*Figure 11-3. Course objectives*

## 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 2016

*Figure 11-4. Enhance your learning with IBM resources*



IBM

## Course completion

You have completed this course:

- Develop, Publish, Manage, and Secure APIs with IBM API Connect V5.0

Any questions?



[Course summary](#)

© Copyright IBM Corporation 2016

*Figure 11-5. Course completion*



IBM Training



© Copyright International Business Machines Corporation 2016.