



Global Knowledge®

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

Course Exercises Guide

WD514G, ERC: 1.1
100402, Version 001
wd51411exercises



Global Knowledge®



Global Knowledge®

Course Exercises Guide

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

Course code WD514 / ZD514 ERC 1.1



May 2019 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.

© Copyright International Business Machines Corporation 2019.

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

Trademarks	v
Exercises description	11
Exercise 1. Review the API Connect development and runtime environment	1-1
1.1. Review the network connectivity and domains	1-3
1.2. Review the Kubernetes runtime environment	1-5
1.3. Review resources in Cloud Manager	1-9
Exercise 2. Create an API definition from an existing API	2-1
2.1. Review the Saving Plan REST API sample application	2-4
2.2. Create an API definition	2-9
2.3. Invoke a GET operation in the existing API implementation	2-19
2.4. Test the API operation in the assembly	2-22
2.5. Define a POST API operation with a JSON request message	2-28
2.6. Test the savings API with the assembly test feature	2-34
Exercise 3. Define an API that calls an existing SOAP service	3-1
3.1. Review the existing SOAP web service	3-4
3.2. Create a SOAP API definition from a WSDL document	3-8
3.3. Test the API on the DataPower gateway	3-16
Exercise 4. Create a LoopBack application	4-1
4.1. Create a LoopBack application with the apic command line utility	4-3
4.2. Examine the structure of a LoopBack application	4-4
4.3. Import the API definition into the API Manager web application	4-12
4.4. Review the API definition in the API Manager web application	4-15
4.5. Test the API operation with the LoopBack API Explorer	4-21
Exercise 5. Define LoopBack data sources	5-1
5.1. Create the inventory application and MySQL data source	5-4
5.2. Create the item LoopBack model	5-7
5.3. Test the inventory application and items model	5-12
5.4. Create a MongoDB data source and the review model	5-16
5.5. Define a relationship between the item and review models	5-19
5.6. Test the inventory application and review model	5-22
Exercise 6. Implement event-driven functions with remote and operation hooks	6-1
6.1. Modify the API base path	6-4
6.2. Create an operation hook	6-5
6.3. Test the operation hook	6-8
6.4. Create a remote hook	6-10
6.5. Test the remote hook	6-17
Exercise 7. Assemble message processing policies	7-1
7.1. Import the inventory API definition into API Manager	7-4
7.2. Test the inventory API definition by calling it on the gateway	7-11
7.3. Create the financing API definition	7-16
7.4. Create an API operation in the financing API	7-21

7.5. Set activity logging in the financing API	7-24
7.6. Invoke a SOAP web service with a message processing policy	7-26
7.7. Test the financing APIs by calling it on the gateway	7-35
7.8. Create the logistics API definition	7-39
7.9. Edit the logistics API definition	7-41
7.10. Define the message policies for the GET /stores API operation	7-44
7.11. Test the stores API by calling them on the gateway	7-49
Exercise 8. Declare an OAuth 2.0 provider and security requirement	8-1
8.1. Create a Native OAuth Provider	8-4
8.2. Configure OAuth 2.0 authorization in the inventory API	8-15
8.3. Create a test application	8-18
8.4. Test OAuth security in the inventory API	8-20
Exercise 9. Deploy an API implementation to a container runtime environment	9-1
9.1. Test a local copy of the LoopBack API application	9-4
9.2. Set up the Docker image configuration	9-6
Exercise 10. Define and publish an API product	10-1
10.1. Create a product for all three APIs	10-4
10.2. Verify that the target URL in the inventory API routes to the Docker image	10-12
10.3. Enable API key security in the financing and logistics API	10-14
10.4. Publish the product and API definitions	10-16
Exercise 11. Subscribe and test APIs	11-1
11.1. Review the consumer organizations in the Sandbox catalog	11-4
11.2. Register a client application in the Developer Portal	11-7
11.3. Subscribe the think application to a product and plan	11-12
11.4. Test subscribed APIs in the Developer Portal test client	11-17
11.5. Update the client application with the client ID and client secret	11-27
11.6. Test all the APIs with the consumer client web application	11-29
Exercise 12. Troubleshooting the case study - optional	12-1
12.1. Review the inventory items API operation	12-3
12.2. Test the OAuth authorize and token API operations	12-4
12.3. Verify the OAuth 2.0 Provider API configuration	12-10
12.4. Test the inventory items API operation	12-11
Exercise 13. Assemble the shipping message processing policy - optional	13-1
13.1. Review and change the logistics API definition	13-4
13.2. Define message processing policy for the logistics API shipping operation	13-8
13.3. Publish the API and start the back-end application	13-15
13.4. Test the logistics API policy assembly on the Developer Portal	13-17
13.5. Test the logistics GET /shipping API with the consumer client web application	13-21
Appendix A. Solutions	A-1
Appendix B. General troubleshooting issues when working on the exercises	B-1
B.1. Useful Kubernetes information and commands	B-7

Trademarks

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

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

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

Bluemix®

DB™

IBM Bluemix™

Cloudant®

Express®

IMS™

DataPower®

IBM API Connect™

Notes®

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

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

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

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

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

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

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



Exercises description

This course includes the following exercises:

- [Exercise 1, "Review the API Connect development and runtime environment"](#)
- [Exercise 2, "Create an API definition from an existing API"](#)
- [Exercise 3, "Define an API that calls an existing SOAP service"](#)
- [Exercise 4, "Create a LoopBack application"](#)
- [Exercise 5, "Define LoopBack data sources"](#)
- [Exercise 6, "Implement event-driven functions with remote and operation hooks"](#)
- [Exercise 7, "Assemble message processing policies"](#)
- [Exercise 8, "Declare an OAuth 2.0 provider and security requirement"](#)
- [Exercise 9, "Deploy an API implementation to a container runtime environment"](#)
- [Exercise 10, "Define and publish an API product"](#)
- [Exercise 11, "Subscribe and test APIs"](#)
- [Exercise 12, "Troubleshooting the case study - optional"](#)
- [Exercise 13, "Assemble the shipping message processing policy - optional"](#)
- [Appendix A, "Solutions"](#)
- [Appendix B, "General troubleshooting issues when working on the exercises"](#)

In the exercise instructions, you can check off the line before each step as you complete it to track your progress.



Hint

If you are unable to complete an exercise, you can copy the model solution application from the **lab files** directory (/home/localuser/lab_files). See [Appendix A, "Solutions"](#) for instructions on how to access the solution code and application for each exercise.



Important

Online course material updates might exist for this course. To check for updates, see the Instructor wiki at: <http://ibm.biz/CloudEduCourses>

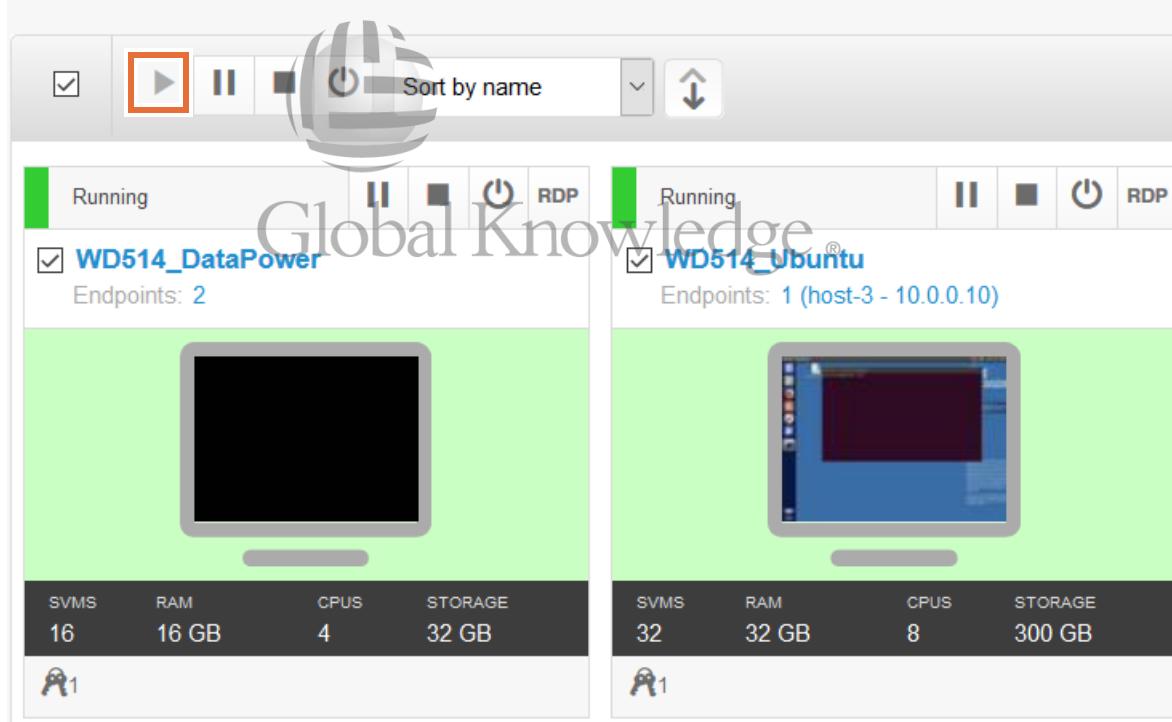
Before you begin

You complete the instructions in this exercise on the remote lab environment. Before you begin your exercise, make sure that the virtual machines for the IBM API Connect cloud are running.

Leave the virtual machines running while you are busy working on the class exercises. When you are finished for the day, shut down both images by clicking the square shutdown icon. The system might suspend the lab environment after some time to preserve resources and to save the current state of the virtual machines.

- ___ 1. In the IBM Remote Lab Platform, make sure that the virtual machines are started.
 - ___ a. Examine the connection to the IBM Remote Lab Platform for the virtual machines that are used in this course.
 - ___ b. There are 2 image virtual machines for this class:
 - WD514 Student Ubuntu image
 - WD514 DataPower
 - ___ c. If the course image virtual machines are not started, click the start arrow for the environment to start the virtual machines.
 - ___ d. The background color changes to green and the status changes to “Running”.

VMs: 2



- ___ 2. Open the student workstation on the Ubuntu Host virtual machine.
 - ___ a. Click the picture of the desktop in the running course image pane.

- ___ b. A remote desktop connection opens in a web browser window. Wait until the connection opens.

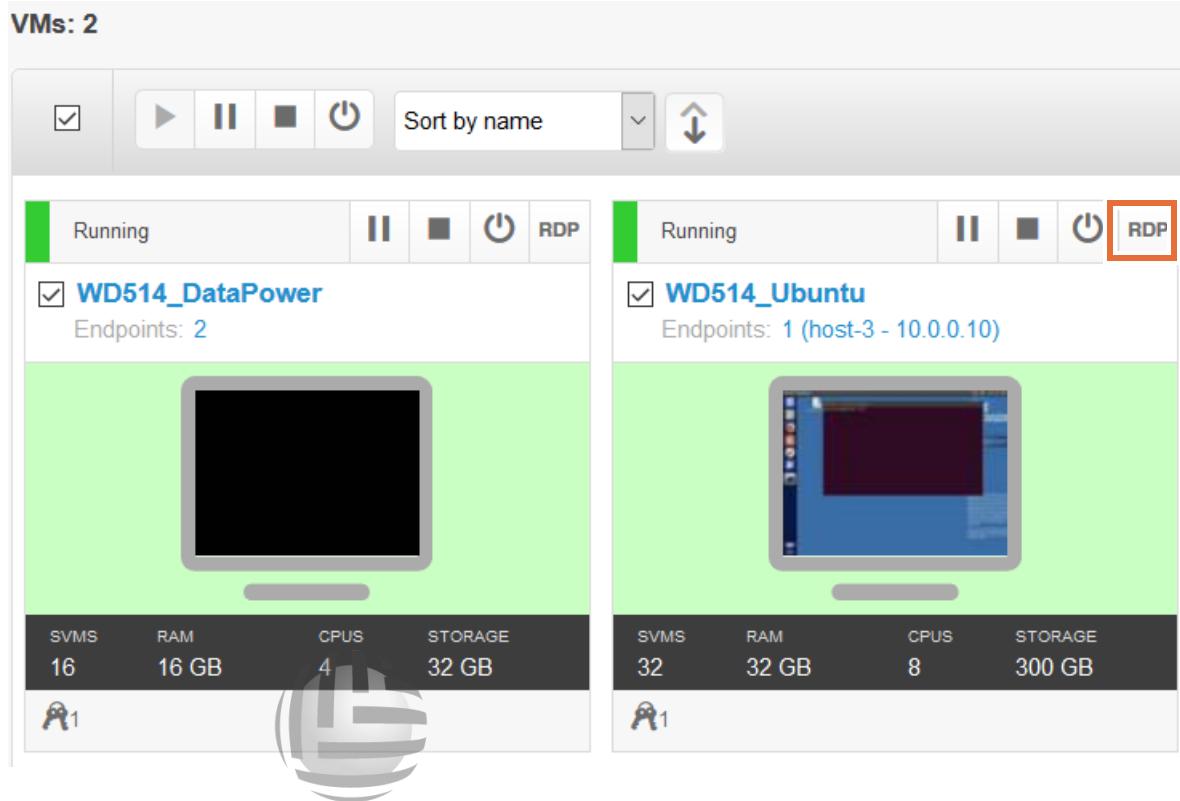


Information

The Ubuntu student workstation opens to the desktop. If you need to sign on to the image, or if you shutdown and restart the Ubuntu server, you can sign on with the credentials:

- User: localuser
- Password: passw0rd

3. You can access the course images either with a web browser or by using a remote desktop connection. If you want to access the student image with a remote desktop connection, click the **RDP** icon on the Ubuntu image and download the RDP file, or open it with a remote desktop application that is installed on your own workstation.

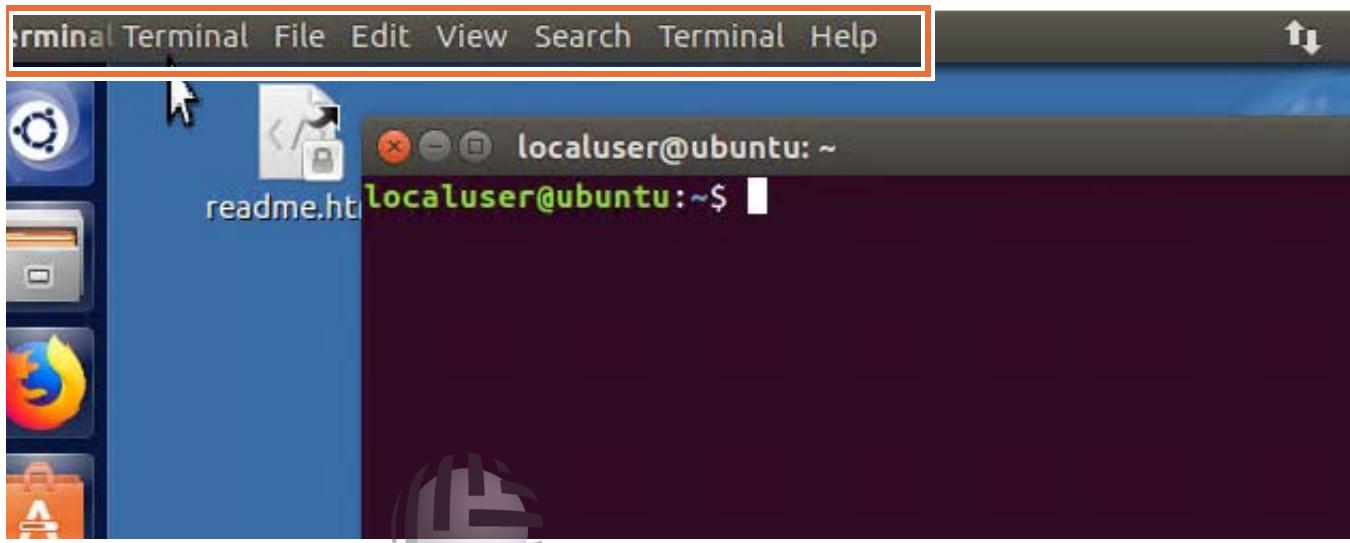


Global Knowledge ®

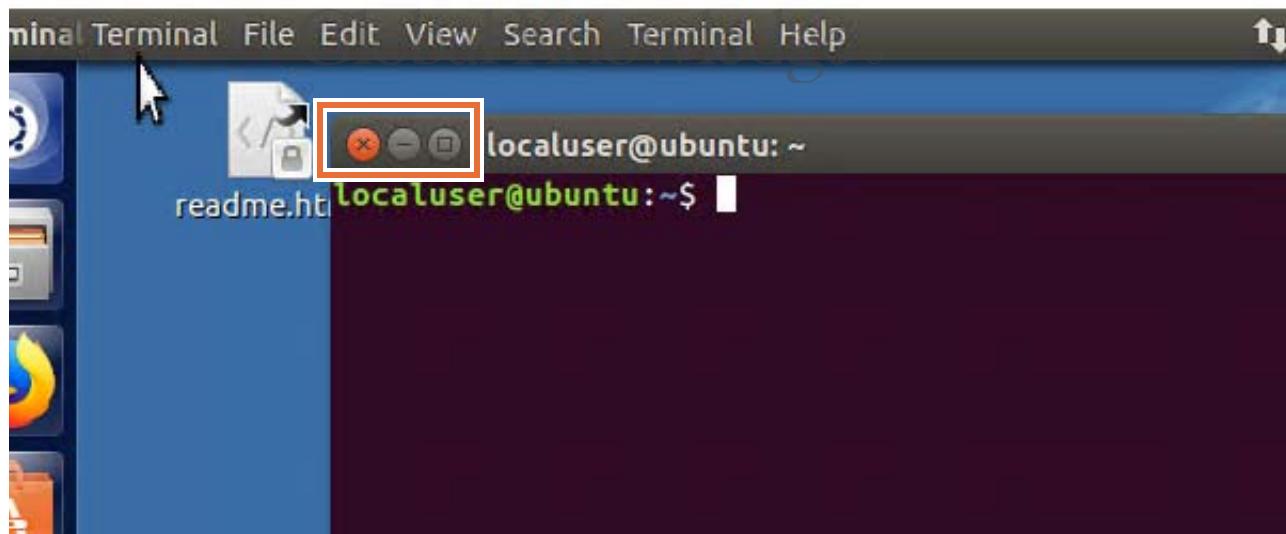


Information

The supplied course image is Ubuntu 16.04 LTS. The desktop for Ubuntu is the Unity desktop. Unity uses a global menu. Which means application menus are not located in the window for the application. They are on the top pane. When a window is the active window, that window does not have any menu items, but the application type is displayed in the black bar that spans the top of the desktop. You cannot see the menu for the application until you hover your mouse over the top pane. When you hover your move over the black bar, the menu items for the active window are displayed.



In the corner of the application, you have the close, minimize, and full screen options.



When you maximize the application window, the close, minimize, and full screen options also appear in the top pane.

Course exercise files

The exercises in this course use a set of lab files that might include scripts, applications, files, solution files, and others. The course lab files can be found in the following directory:

- `/home/localuser/lab_files` for the Ubuntu operating system

The exercises point you to the lab files as you need them.





Global Knowledge®

Exercise 1. Review the API Connect development and runtime environment

Estimated time

00:45

Overview

In the first part of the exercise, you test that you can access the Internet and that your private domain name service is working. You review and validate that the Kubernetes runtime environment and API Connect processes are running. Then, you sign on as the administrator to the Cloud Manager user interface. Review the services that are defined in the Cloud Manager. Review the provider organization that publishes the APIs.

Objectives

After completing this exercise, you should be able to:

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

Requirements

This exercise requires a workstation with internet access. You can complete this exercise by using the Ubuntu course image that is supplied with the course.

The image on the IBM Remote Platform requires these resources:

- 8 CPUs
- 32 GB RAM
- 300 GB HDD

The image might not start all the processes properly when these resources are not configured.

Exercise instructions

Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.

As an API Developer, you might not concern yourself with how to set up the API Connect environment. In this class, you do most of the API development on the remote student image where all the API Connect components are already installed and configured.

However, this exercise gives you some background on the API Connect development and runtime environment that is used in this course. This information can be useful in situations where API administration and development functions overlap and for troubleshooting issues that you might have when running the exercises on the course image.

This first exercise also verifies that your course environment is working properly, since all the exercises run from the course image.



1.1. Review the network connectivity and domains

In this part, you validate your connectivity to the internet and the function of the private DNS.

The network on the student image is configured as a private DNS server on Ubuntu with the BIND package. In the exercises, you use the `think.ibm` domain that is configured on the primary DNS server, which is the student image itself (IP address 10.0.0.10).

- ___ 1. The network connectivity and naming lookup.
 - ___ a. Open a terminal from the Ubuntu desktop launcher.



- ___ b. In the terminal, type `nslookup google.com`.
The result is displayed.

```
localuser@ubuntu:~$ nslookup google.com
Server:10.0.0.10
Address:10.0.0.10#53
```

Non-authoritative answer:
Name:google.com
Address: 172.217.14.206

- c. In the terminal, type nslookup cloud.think.ibm.
The result is displayed.

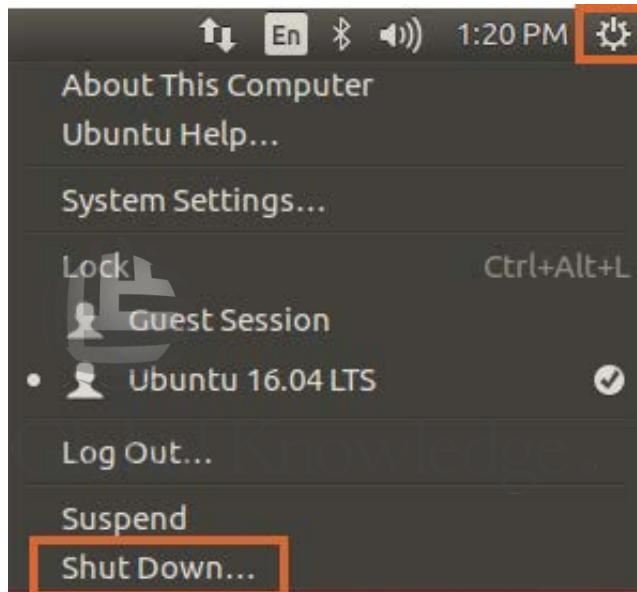
```
localuser@ubuntu:~$ nslookup cloud.think.ibm
Server:10.0.0.10
Address:10.0.0.10#53

Name:cloud.think.ibm
Address: 10.0.0.10
```



Important

If these queries do not work, check whether you can access the internet from the image. Restart the image by selecting shutdown from the menu.



Then, select Restart. When the image restarts, retest the connectivity and naming lookup.

1.2. Review the Kubernetes runtime environment

For this course, IBM API Connect runs on a Kubernetes environment, sometimes abbreviated as K8. The Kubernetes environment supports scalability and failover. For this course, Kubernetes is set up with a single master node and all the processes run on the same virtual machine. This configuration is not scalable and is used only for demonstration purposes. Kubernetes manages the Docker containers that provide the runtime environment. These components start automatically when the student image is started. You might need to wait up to 10 minutes after you sign on to the student image for all the processes to start and for the API Connect environment to become fully operational.

- 1. Review the Helm charts that are defined on the image.

Helm is the Kubernetes package manager.

- a. Click the Terminal in the list of applications on the Ubuntu desktop.

- b. Type `helm list`.

The list of deployed helm charts is displayed.

NAME	REVISION	UPDATED	STATUS	CHART	APP
VERSION	NAME	SPACE			
ingress	1		Mon Jan 14 19:54:27 2019	DEPLOYEDnginx-ingress-1.1.2	0.21.0
apiconnect					
r2484482d491			Mon Jan 14 20:36:14 2019	DEPLOYEDapic-portal-2.0.0	
apiconnect					
r674f0bc86d1			Mon Jan 14 20:03:40 2019	DEPLOYEDapiconnect-2.0.0	
apiconnect					
r8e789c134d1			Mon Jan 14 20:03:30 2019	DEPLOYEDcassandra-operator-1.0.01.0.1	
apiconnect					
re266d799751			Mon Jan 14 20:14:56 2019	DEPLOYEDapic-analytics-2.0.0	
apiconnect					



Note

If you get a message: Error: could not find tiller

Type: `export TILLER_NAMESPACE=apiconnect`

In the terminal.

Retry the `helm list` command.

- 2. Display the pods that are running on the apiconnect namespace.

- a. In the Terminal type:

```
kubectl get pods -n apiconnect
```

b. The list of pods is displayed.

NAME	READY	STATUS
RESTARTS	AGE	
hostpath-provisioner-77d68bd579-sfltx	1/1	Running 3
41h		
ingress-nginx-ingress-controller-m8gr2	1/1	Running 6
41h		
ingress-nginx-ingress-default-backend-7f7bf55777-bldrw	1/1	Running 3
41h		
r2484482d49-apic-portal-db-0	2/2	Running 6
40h		
r2484482d49-apic-portal-nginx-7c4f57cb9-4hs16	1/1	Running 3
40h		
r2484482d49-apic-portal-www-0	2/2	Running 10
40h		
r674f0bc86d-a7s-proxy-76cbb9ddb8-ggkrx	1/1	Running 3
41h		
r674f0bc86d-apiconnect-cc-0	1/1	Running 3
41h		
r674f0bc86d-apiconnect-cc-cassandra-stats-1547663400-nwnb9	0/1	Completed 0
179m		
r674f0bc86d-apiconnect-cc-cassandra-stats-1547667000-h96vp	0/1	Completed 0
119m		
r674f0bc86d-apiconnect-cc-cassandra-stats-1547670600-j6tlw	0/1	Completed 0
58m		
r674f0bc86d-apim-schema-init-job-w4xgp	0/1	Completed 0
41h		
r674f0bc86d-apim-v2-5447b859f8-gnk8v	0/1	Running 3
41h		
r674f0bc86d-client-dl-srv-75ff766974-gz2jg	1/1	Running 3
41h		
r674f0bc86d-juhu-6c8cb7f88b-v8cff	1/1	Running 3
41h		
r674f0bc86d-ldap-75fc745cdc-6zqrs	1/1	Running 4
41h		
r674f0bc86d-lur-schema-init-job-cp2gt	0/1	Completed 0
41h		
r674f0bc86d-lur-v2-fdd4479dd-rw415	1/1	Running 4
41h		
r674f0bc86d-ui-66697bdb7b-62561	1/1	Running 3
41h		
r8e789c134d-cassandra-operator-554547cc77-jpxvz	1/1	Running 3
41h		
re266d79975-analytics-client-5964d5bd66-1xkbx	1/1	Running 3
41h		
re266d79975-analytics-cronjobs-rollover-1547673300-rxz79	0/1	Completed 0
14m		
re266d79975-analytics-ingestion-5b6d9dfb54-2qfp1	1/1	Running 4

```

41h
re266d79975-analytics-mtls-gw-575d8f644f-h2k5q           1/1   Running   3
41h
re266d79975-analytics-storage-coordinating-65d58c8b8d-bqgr4 1/1   Running   6
41h
re266d79975-analytics-storage-data-0                     1/1   Running   6
41h
re266d79975-analytics-storage-master-0                  1/1   Running   6
41h
tiller-deploy-7d5db86b78-twbgx                         1/1   Running   3
41h

```

- c. Most of the pods should have a status of "Running" or "Completed" as shown. If some of the pods are still initializing, you can reissue the command. It is acceptable when some of the cassandra-stats pods have a status of error.
 - d. The system is ready to run API Connect.
-



Information

You can also run the command `docker ps`. The system can sometimes take about 10 minutes to start all the docker containers.

Other commands that you can type to check the status of the Kubernetes runtime environment, are:

```

localuser@ubuntu:~$ kubectl get nodes
NAME      STATUS    ROLES     AGE      VERSION
ubuntu    Ready     master    42h     v1.13.1

```

The output from running this command shows a master node and the Kubernetes version.

```

localuser@ubuntu:~$ kubectl get pods -n kube-system
NAME                           READY   STATUS    RESTARTS   AGE
calico-etcd-nxs7s              1/1     Running   3          42h
calico-kube-controllers-5d94b577bb-crkxh 1/1     Running   5          42h
calico-node-16w21               1/1     Running   3          42h
coredns-86c58d9df4-6lft4       1/1     Running   3          42h
coredns-86c58d9df4-ngnhl       1/1     Running   3          42h
etcd-ubuntu                     1/1     Running   3          42h
kube-apiserver-ubuntu          1/1     Running   3          42h
kube-controller-manager-ubuntu  1/1     Running   4          42h
kube-proxy-hwlk4                1/1     Running   3          42h

```

kube-scheduler-ubuntu

1 / 1

Running

4

42h

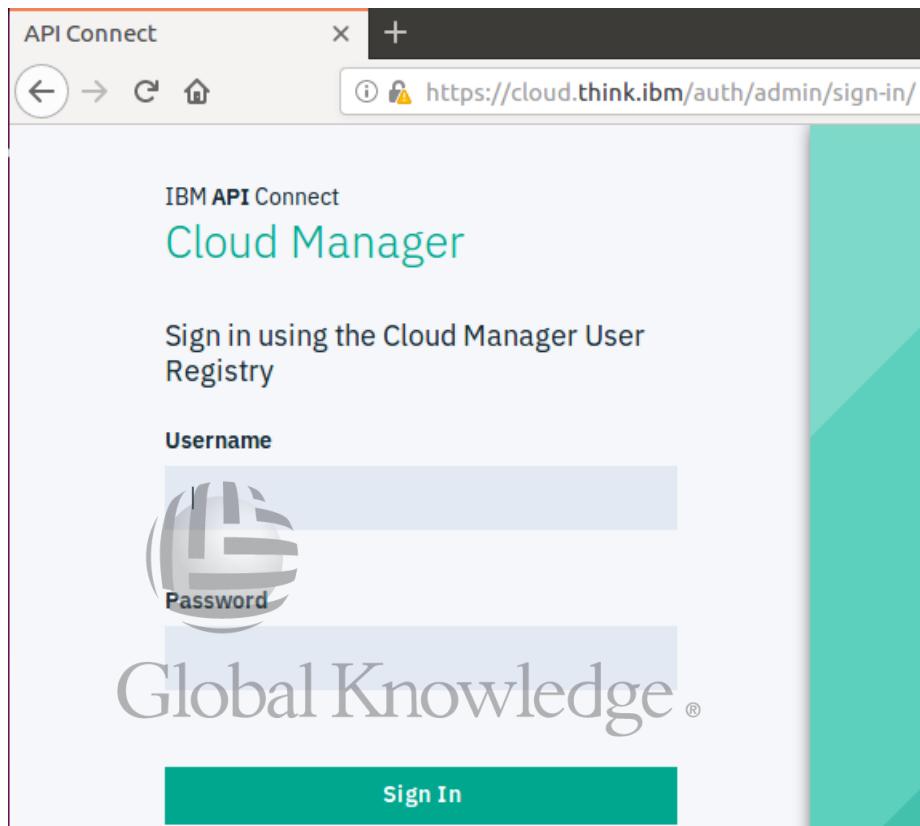


Global Knowledge ®

1.3. Review resources in Cloud Manager

In this part, you review the resources and services that are defined in the Cloud Manager web interface of API Connect.

- 1. Open the Cloud Manager in a browser.
 - a. In the Firefox browser, type `https://cloud.think.ibm` in the address area of the browser.



Note

If the Cloud Manager page returns an API Error, you might need to wait a while longer for the API Connect environment on Kubernetes to start. Environment initialization might take about 10 minutes.

-
- b. Sign on to Cloud Manager with the administrator credentials:
 - User name: admin
 - Password: Passw0rd!
- The user is signed in to Cloud Manager.
-

- __ 2. Review the services that are defined in Cloud Manager.
 - __ a. Click the **Configure Topology** tile or **Topology** from the navigation menu.

The image shows the 'Welcome to the Cloud Manager' screen of the IBM API Connect interface. On the left, there is a vertical sidebar with several icons: a gear, a cluster, a gear with a bar chart, a gear with a gear, a clipboard, and a gear with a cloud. The icon for 'Configure Cloud' is highlighted with a red border. The main area features a large title 'Welcome to the Cloud Manager' and a subtitle 'Choose an option to get started'. Below this, there are two main sections: 'Configure Cloud' (with a gear and cloud icon) and 'Configure Topology' (with a line graph and a 'no' symbol icon). Each section has a brief description below it.

Welcome to the Cloud Manager

Choose an option to get started

Configure Cloud

Edit settings for user registries, roles, endpoints, and more

Configure Topology

Manage availability zones and services

The topology page is displayed. A Management service is already configured in the default availability zone when API Connect is installed.

Default Availability Zone		Management	Register Service
Register new services and manage existing services			
Service	Type	Associated Analytics Service	Visible To
 API Datapower Gateway	DataPower API Gateway	analytics-service	Public
 Portal Service	Portal Service		Public
 Analytics Service	Analytics Service		

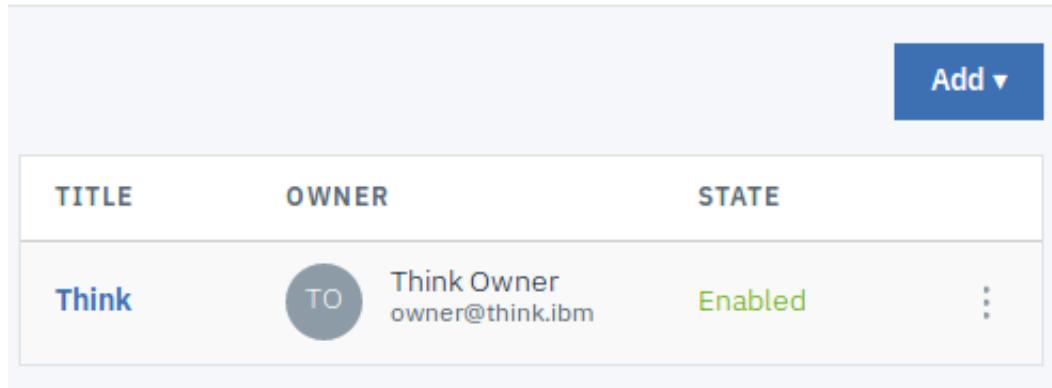
A gateway service, portal service, and analytics service are already configured.

- __ 3. Review the provider organization that is already defined.

a. Click **Provider Organizations** in the left navigation bar.

- ___ b. A provider organization that is named Think is displayed in the list of provider organizations. The owner of the provider organization is Think Owner.

Provider Organizations



TITLE	OWNER	STATE	
Think	TO Think Owner owner@think.ibm	Enabled	:

In later exercises, you sign on to the API Manager user interface with the credentials of Think Owner.

- ___ 4. Review the user registries.
 - ___ a. Click **Resources** in the left navigation bar.
 - ___ b. You see the two local user registries that are defined.
- | <input type="checkbox"/> | TITLE | TYPE | VISIBLE TO |
|--------------------------|-----------------------------------|---------------------|------------|
| <input type="checkbox"/> | API Manager Local User Registry | Local User Registry | Private |
| <input type="checkbox"/> | Cloud Manager Local User Registry | Local User Registry | Private |
- If you do not define user registries for your organization, API Connect uses the default local user registries.
 - ___ c. You cannot directly open the local user registries to view the users. However, you can indirectly query members in the local user registries, as you see in a later exercise.
 - ___ d. The admin user was created in the Cloud Manager local user registry during product installation.
 - ___ e. The ThinkOwner user was created in the API Manager local user registry when the Think provider organization was added.
- ___ 5. Review the Cloud Manager settings.
 - ___ a. Click **Cloud Settings** in the left navigation bar.

- ___ b. On the Settings page, click **Role Defaults**. You see a list of the predefined roles for Provider Organizations in API Connect.

Provider Organization

Configure the set of roles to use by default when a provider organization is created

ROLES

- > Administrator
- > API Administrator
- > Community Manager
- > Developer
- > Member
- > Owner
- > Viewer



- ___ c. Expand **Developer** to see the permissions for the role of an API Developer in the provider organization.
- ___ d. In many circumstances, users are created with the Developer role that can be used for creating and editing APIs.

- __ e. Expand **Owner** to see the permissions of the owner of the provider organization.

Owns and administers the API provider organization		
Member	Settings	Topology
<ul style="list-style-type: none"> • View • Manage 	<ul style="list-style-type: none"> • View • Manage 	<ul style="list-style-type: none"> • View • Manage
Org <ul style="list-style-type: none"> • View 	Drafts <ul style="list-style-type: none"> • View • Edit 	Product <ul style="list-style-type: none"> • View • Stage • Manage
Product-Approval <ul style="list-style-type: none"> • View • Stage • Publish • Supersede • Replace • Deprecate • Retire • Archive 	Consumer-Org <ul style="list-style-type: none"> • View • Manage 	App <ul style="list-style-type: none"> • View • Manage
App-Dev <ul style="list-style-type: none"> • Manage 	App-Approval <ul style="list-style-type: none"> • View • Manage 	Subscription <ul style="list-style-type: none"> • View • Manage

The Owner has permissions for all functions in the Provider Organization. Since this user is already created, you use the user with the owner role for API development in this course.

- ___ f. Click **Endpoints** from the Settings page.

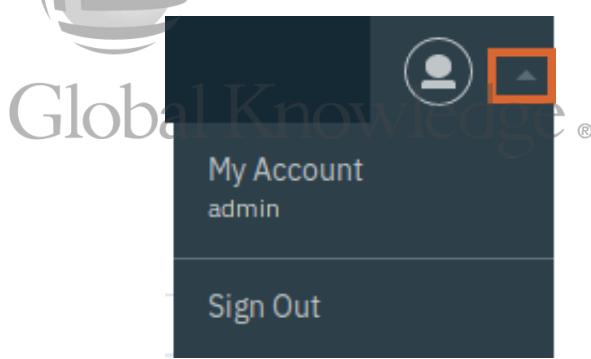
Settings

The screenshot shows the 'Settings' page with a sidebar on the left containing links: Overview, Onboarding, User Registries, Roles, Role Defaults, Endpoints (which is highlighted in blue), and Notifications. The main content area is titled 'Endpoints'. It displays the following information:

- API Manager URL:** <https://manager.think.ibm/manager>
- Platform REST API endpoint for admin and provider APIs:** <https://platform.think.ibm/api>
- Platform REST API endpoint for consumer APIs:** <https://consumer.think.ibm/consumer-api>

The endpoint displays the URL that you use to sign on to the API Manager.

- ___ 6. Sign out of Cloud Manager by selecting the **Sign Out** option from the drop-down menu.



End of exercise

Exercise review and wrap-up

In the exercise, you worked with the IBM API Connect Cloud Manager.

The Cloud Manager is used to define your API Connect topology and Provider Organizations.

In the first part, you reviewed the network connectivity and verified that the private DNS is working.

Next, you looked at some of the Kubernetes pods where API Connect is running.

Finally, you reviewed the services in Cloud Manager and you reviewed the Provider Organization and some of the default role settings in Cloud Manager.





Global Knowledge®

Exercise 2. Create an API definition from an existing API

Estimated time

01:00

Overview

This exercise covers how to define an API interface from an existing API. You review the API operations, parameters, and definitions from the API Manager web application. You also publish and test the API from the API Manager test feature.

Objectives

After completing this exercise, you should be able to:

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

Introduction

You can define APIs with either of these approaches:

- In an **interface-first** design, you define each API path, operation, request, and response message in an OpenAPI definition. You map the interface to an existing API implementation that is deployed in your architecture.
- In an **implementation-first** design, you build an API implementation as a collection of models, properties, relationships, and data sources. You generate a set of REST API operations from the models to an OpenAPI definition.

In this exercise, you build an OpenAPI definition based on the business requirements. The savings API provides a set of financial calculators to help customers estimate the potential return on their savings accounts. After you test a working API implementation, you independently design a set of API operations in the API Manager. You map the operations to the API implementation with an invoke message policy. Last, you test the API definition with the built-in test client of API Manager.

Requirements

You must complete this lab exercise in the student development workstation: the Ubuntu host environment. This virtual machine is preconfigured with the API Connect components so that you can create, edit, and test APIs.



Exercise instructions

Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



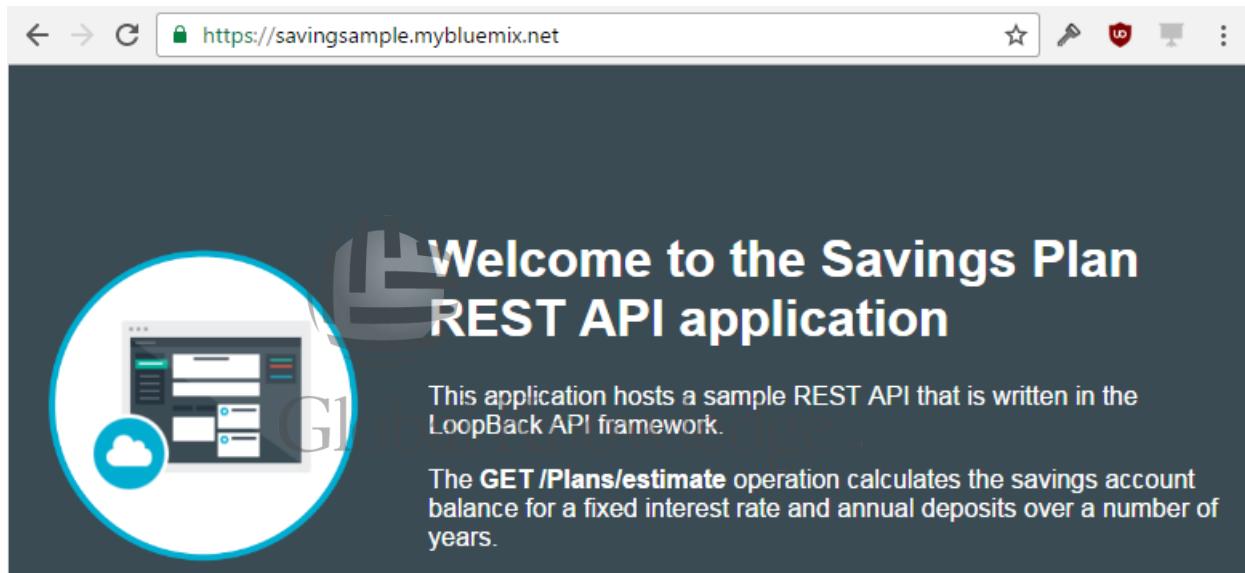
Global Knowledge®

2.1. Review the Savings Plan REST API sample application

The **Savings Plan** REST API projects a savings account balance. The GET /Plans/estimate operation calculates the savings account balance after several years of investment at a fixed interest rate. The operation assumes that you deposit the same amount at the end of each year.

In this section, review the operation of the Savings Plan API. Examine the HTTP request parameters for the API operation, and the structure of the API response message. You use this information to map your API definition to this service that runs on the IBM Cloud.

- ___ 1. Open the Savings Plan REST API site.
 - ___ a. Open a web browser from the student image. Then, type <http://savingsample.mybluemix.net> to open the site.



- ___ 2. Test the GET /Plans/estimate API operation.
 - ___ a. Scroll down the page.
 - ___ b. In the test client, enter the following values:
 - Deposit: 300
 - Interest rate: 0.04 (to represent 4%)

- Years to invest: 20

How much will you deposit at the end of each year?

What is the annual interest rate?

How many years do you want to invest your savings?

Calculate

- ___ c. Click **Calculate**.
- ___ d. Review the results.

The balance is 8933.42 after investing 300 per year over 20 years at an annual interest rate of 4%.

[View API response message](#)



Information



The web page calls the REST API operations with the deposit, rate, and number of years that you typed into the form. The page takes the response from the API call and writes a result on the page.

In this example, you deposited \$300 into a savings account at the end of each year for 20 years. The account earns an annual interest rate of 4%, compounded annually. At the end of 20 years, the savings account balance is \$8,933.42.

-
- ___ 3. Review the raw HTTP request and response messages for the GET /Plans/estimate API operation.
 - ___ a. Select **View API response message**.

- __ b. Examine the API request message source.

The screenshot shows a browser window with the URL <https://savingsample.mybluemix.net/api/Plans/estimate?deposit=3008>. The browser interface includes back, forward, and home buttons, a search bar, and tabs for JSON, Raw Data, and Headers. Below the tabs are Save, Copy, and Pretty Print buttons. The main content area displays a single JSON object: {"balance": 8933.42}.



Information

The network path for the API operation is

`http://savingsample.mybluemix.net/api/Plans/estimate`. The operation expects the input parameters as query parameters. As a GET method call, the HTTP request message body is empty. The API returns a response message as a JSON object.

- __ 4. Test the `GET /Plans/estimate` API operation from your workstation with the cURL utility.

- __ a. Open a terminal application window.
- __ b. Make an HTTP GET request to the savings sample.

```
$ curl -v  
"https://savingsample.mybluemix.net/api/Plans/estimate?deposit=300&rate=0  
.04&years=20"
```

___ c. Examine the result from the API operation call.

```
* Connected to savingsample.mybluemix.net (169.47.124.22) port 443 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
* found 594 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_RSA_AES_256_GCM_SHA384
*   server certificate verification OK
*   server certificate status verification SKIPPED
*   common name: *.mybluemix.net (matched)
*   server certificate expiration date OK
*   server certificate activation date OK
*   certificate public key: RSA
*   certificate version: #3
*   subject: C=US,ST>New York,L=Armonk,O=International Business Machines
Corporation,CN=*.mybluemix.net
*   start date: Thu, 13 Apr 2017 00:00:00 GMT
*   expire date: Fri, 10 Jul 2020 12:00:00 GMT
*   issuer: C=US,O=DigiCert Inc,CN=DigiCert SHA2 Secure Server CA
*   compression: NULL
* ALPN, server did not agree to a protocol
> GET /api/Plans/estimate?deposit=300&rate=0.04&years=20 HTTP/1.1
> Host: savingsample.mybluemix.net
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Backside-Transport: OK OK
< Connection: Keep-Alive
< Transfer-Encoding: chunked
< Access-Control-Allow-Credentials: true
< Content-Type: application/json; charset=utf-8
< Date: Wed, 16 Jan 2019 23:41:42 GMT
< Etag: W/"13-HMDcdMEKZozomevxPCCDCLbJFCE"
< Vary: Origin, Accept-Encoding
< X-Content-Type-Options: nosniff
< X-Download-Options: noopen
< X-Frame-Options: DENY
< X-Xss-Protection: 1; mode=block
< X-Global-Transaction-ID: 1969289405
<
* Connection #0 to host savingsample.mybluemix.net left intact
{ "balance":8933.42}
```



Information

The cURL utility is a widely customizable, third-party application for HTTP network testing. In this example, you sent an HTTP GET request to the GET /Plans/estimate API operation. The **-v** verbose parameter displays the headers in the HTTP request, and the headers and body from the HTTP response message.

In this course, you use a combination of command line utilities and web application clients to test API operations.



2.2. Create an API definition

API Connect provides two platforms to develop APIs: the API Designer and API Manager.

The API Designer is a workstation-based, graphical development environment.

When working with the API Designer, you normally create a directory on the development workstation for the application that contains the API definition.

API Manager is a web-based application for the development and management of APIs.

In API Manager, APIs and Products are displayed in the Develop page of the web application and API Manager manages these artifacts internally. You can download and save the YAML files that are produced in API Manager.



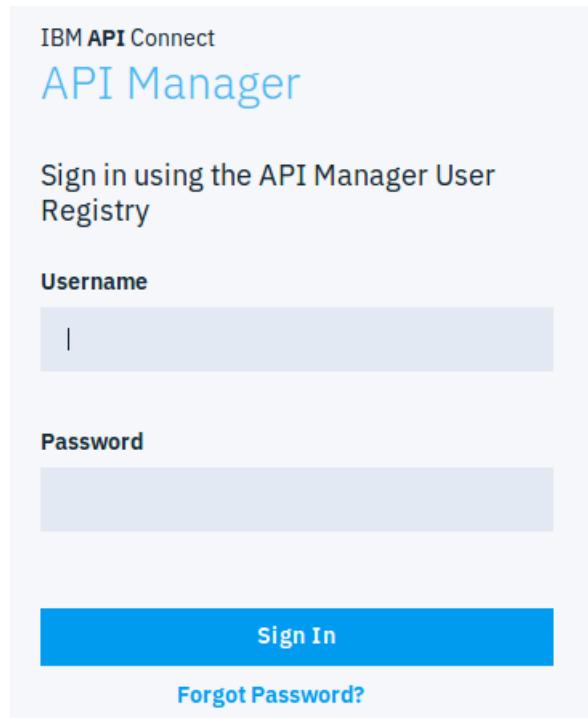
Information

In this course, you use the API Manager browser-based application that runs on the student image for the development of APIs. The graphical interfaces to develop APIs and products are nearly identical in the API Designer and API Manager.

The API Connect components for development are set up on the student image for seamless integration from API Manager. These components include a Sandbox catalog and Sandbox Developer Portal and a separate DataPower image that contains some services that are used in some of the later exercises.

-
- 
- 1. Sign on to the API Manager development environment. You can use the bookmark for API Manager in the Firefox browser.
 - a. Sign on to the API Manager web interface with the host name of the management server from a browser session. Type the address
`https://manager.think.ibm.`

- __ b. Sign in with the user name and password that is associated with your API Manager account.



The image shows the IBM API Connect API Manager sign-in page. The title bar says "IBM API Connect API Manager". Below it, a heading says "Sign in using the API Manager User Registry". There are two input fields: "Username" and "Password", both currently empty. A blue "Sign In" button is at the bottom, and a "Forgot Password?" link is below it. To the right of the input fields is a watermark logo for "Global Knowledge".

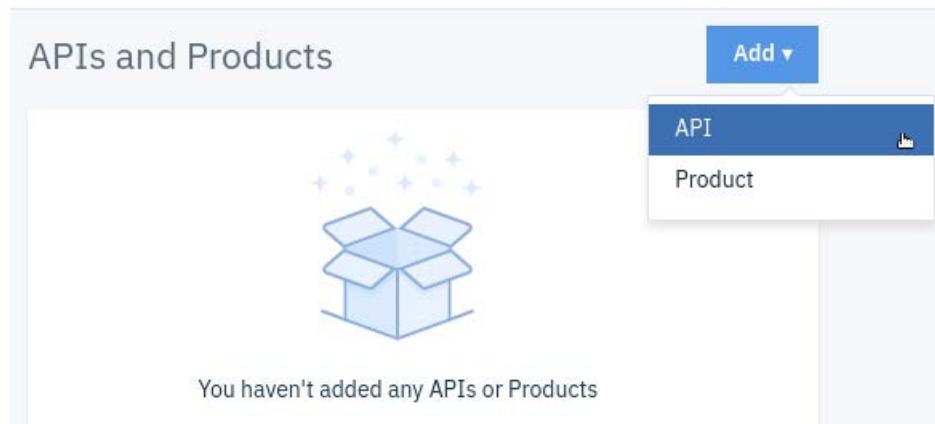
- User name: ThinkOwner
- Password: Passw0rd!

Click **Sign in**.

The user is signed in to API Manager.

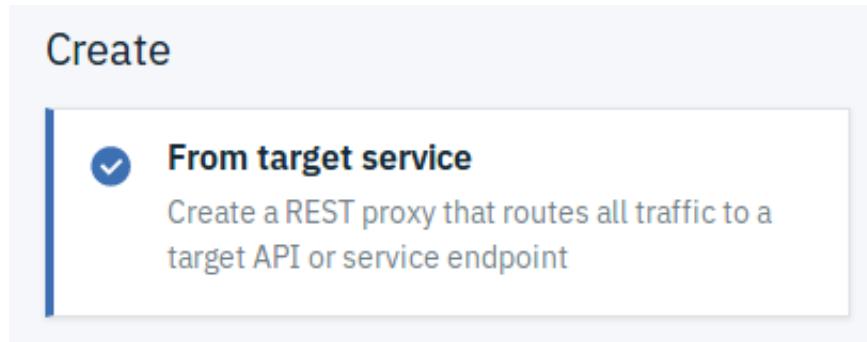
- __ 2. Create an OpenAPI API definition.
- __ a. From the API Manager home page, click the **Develop APIs and Products** tile, or click **Develop** from the navigation bar.
- __ b. From the Develop page, click the **Add** icon. Then, select **API**.

Develop



The image shows the "Develop" page in the IBM API Connect interface. The main title is "APIs and Products". On the left is a large blue icon of a box with stars above it. Below the icon is the text "You haven't added any APIs or Products". On the right, there is a blue "Add" button with a dropdown arrow. A small menu is open, showing "API" and "Product" options. The background has a light gray grid pattern.

- ___ c. Click the tile **From target service**.



The selector is displayed.

- ___ d. Click **Next**.

- ___ e. Type the following properties for the API definition:

- Title: savings
- Name: savings
- Version: 1.0.0
- Base path: /savings
- Target service URL: `http://savingsample.mybluemix.net/api/Plans`

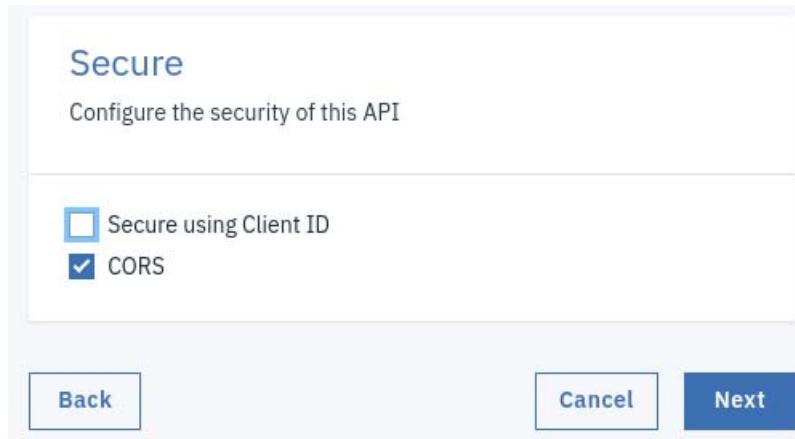
The screenshot shows the 'Create' interface with the following fields filled in:

- Version**: 1.0.0
- Base path (optional)**: /savings
- Description (optional)**: (empty)
- Target Service URL**: Enter the URL for the target service you would like to proxy
- Target service URL**: `http://savingsample.mybluemix.net/api/Plans`

At the bottom right are two buttons: **Cancel** and **Next**.

- ___ f. Click **Next**.

- __ g. Clear **Secure using Client ID** in the Secure dialog.



- __ h. Leave the **CORS** property selected.
__ i. Click **Next**.
__ j. The OpenAPI 2.0 definition is generated.



- __ k. Click **Edit API**.
__ 3. Examine the **savings** API definition in the API Editor.
__ a. The page is displayed with the API Setup option that is selected within the Design view.

- ___ b. Review the API description metadata: name, title, and version.

The screenshot shows the 'Info' section of the 'savings' API. The left sidebar has a 'API Setup' tab selected, showing options like Security Definitions, Security, Paths, Definitions, Properties, Categories, and Activity Log. The main area is titled 'Info' with the sub-section 'Title' containing the value 'savings'. Below it is the 'Name' field with the value 'savings'. At the bottom is the 'Version' field with the value '1.0.0'.

- ___ c. Scroll down in the page. Then, examine the **host** and **base path** sections.

The screenshot shows the 'Host' and 'Base Path' sections. The 'Host' section includes a placeholder 'Address (optional)' with a redacted input field. The 'Base Path' section includes a placeholder 'Base path (optional)' with the value '/savings'.



Information

The **host** is the domain name or IP address (IPv4) of the host that serves the API. If the host is not specified, it is assumed to be the network address of the server that hosts the API definition. By default, API Manager uses the network endpoint of the API gateway that represents the API catalog.

The **base path** represents the network path immediately after the host name. For example, the address of the API Gateway is `http://apigw.think.ibm`. In this case, the entry point into the savings API is `http://apigw.think.ibm/savings`.

- ___ d. Examine the **consumes** and **produces** sections.

Change the definition so that `application/json` is selected for both the consumes and produces media types.

Consumes

Consumes (optional)

application/json
 application/xml

 Add media type (optional)

Global Knowledge®

Produces (optional)

application/json
 application/xml

Add media type (optional)



Information

The **consumes** describes the media types that are used in the HTTP request message body and **produces** describes the media types that are used in the HTTP response message body. In this example, the API operations expect the message body in an application/json data format. Conversely, the API operations return response messages in an application/json format as well.

The **consumes** and **produces** settings are the API-wide default values. You can override these settings at the operation level.

- 4. Examine the gateway type.
 - a. Scroll down in the definition to the Gateway Type.
 - b. Ensure that the DataPower API Gateway is selected.



- 5. Click **Save**.
A message is displayed that the API has been updated.



Note

A warning message is displayed indicating that you need to check the proxy policy type in the assembly before you can run this API. When you modify your API definitions to use DataPower API Gateway, you must ensure that the gateway type supports each policy and policy version in the API assembly.

DataPower Gateway (v5 compatible) supports version 1.0.0 of the invoke policy, but DataPower API Gateway requires version 2.0.0.

The proxy policy is provided by the invoke policy in the DataPower API Gateway. You change the policy later in the assembly.

- ___ 6. Create a path that is named /estimate.
 - ___ a. Select the **paths** section.
 - ___ b. Click **Add** to create a path.
 - ___ c. Change the path name to /estimate.

Path name

/estimate

- ___ d. Click **Save**.
- ___ 7. Define a definition for the return data type.
 - ___ a. Select the **Definitions** section of the API editor.
 - ___ b. Click **Add** to create a schema definition.
 - ___ c. Change the name of the schema definition type to savings-result.
 - ___ d. Set the description to Calculated result from savings plan.

Edit definitions

Name	savings-result
Type	object
Description (optional)	Calculated result from savings plan

Properties

Add

- ___ e. Click **Add** to add a property.

- ___ f. In the plan-result definition properties, define the balance property according to the table.

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

Table 1. Plan-result schema type definition

Property name	Description	Type	Example
balance	Account balance	float	8933.42

- ___ g. Click **Save**.
The definition is saved.
- ___ 8. Define a GET operation for the path.
- ___ a. Click **Paths** to display the paths.
 - ___ b. Click the ellipsis three dots alongside paths. Then, click **Edit**.
 - ___ a. Click **Add** alongside Operations. Then, select GET. Click **Add**.
 - ___ b. The GET operation is added.
 - ___ c. Click **Save**.
If necessary, open the paths again with the Edit option.
 - ___ d. Click the ellipsis (three dots) alongside the GET operation. Then, click **Edit**.
- ___ 9. Define three parameters for the GET operation: deposit, rate, and years.
- ___ a. Click **Add** alongside Parameters three times so that three rows of parameters are displayed.

- ___ b. Type the request parameter values according to the table.

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

Table 2. GET /estimate API operation parameters

Name	Located in	Description	Required	Type
deposit	Query	Annual deposits	Yes	float
rate	Query	Interest rate	Yes	float
years	Query	Years of saving	Yes	int 64

- ___ 10. Define the response message from the GET /estimate operation to return the plan-result object.
- ___ c. Click **Add** alongside Response to add a response
- ___ d. In the response area, type 200 in the **status code**, set the **schema** to **savings-result**, and **description** to **200 OK**.

Response		
STATUS CODE	SCHEMA	DESCRIPTION
200	savings-result	200 OK

- ___ 11. Click **Save** to save the changes to the API definition. The API Manager returns to the page with the path option selected and you see a confirmation dialog that the API is saved.

2.3. Invoke a GET operation in the existing API implementation

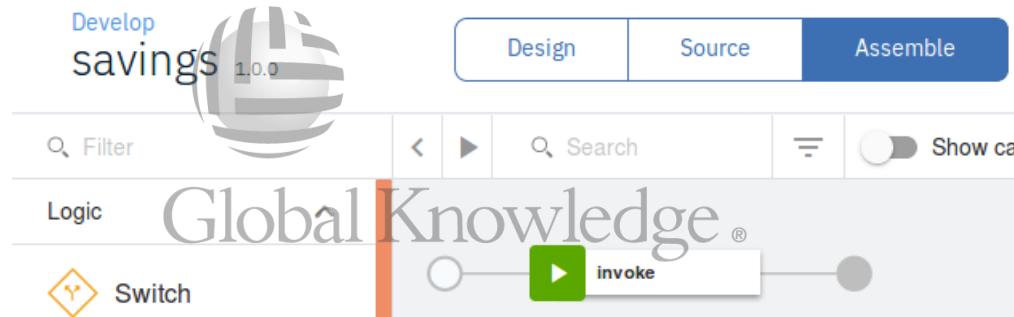
You created an API definition: the interface for the REST API. However, the interface describes the structure of the request and response messages only. It does not implement the actual API operation.

In this section, you send the API operations that you defined in the API definition to the existing Savings Plan REST API application. Modify the **invoke** properties to call the GET /estimate operation in the Savings Plan API. Map the response message to the savings-result object in the savings API definition.

- ___ 1. Open the **Assemble** view in the API editor.
- ___ a. In the savings API definition, click the **Assemble** tab.



- ___ b. Examine the assemble view in the API editor.



- ___ c. Notice that an invoke policy is automatically inserted in the assembly.



Information

What is the **assemble** view?

The assemble view is a graphical editor that defines a sequence of message processing policies. The API Gateway transforms and routes API messages based on these policies. The policies apply to all API operations in the API definition. The sequence in the assembly runs from the left to the right.

-
- ___ 2. Review the properties of the **invoke** policy.
 - ___ a. Double-click the **invoke** policy in the sequence to open the properties view.

- __ b. Examine the **URL** field.

The screenshot shows a configuration screen for an 'invoke' operation. At the top, there is a green play button icon and the word 'invoke'. To the right are icons for edit, add, and delete. Below this, the 'Title' field contains 'invoke'. The 'Description' field is empty. The 'URL *' field contains '\$(target-url)' with a tooltip 'The URL to be invoked.'.



Information

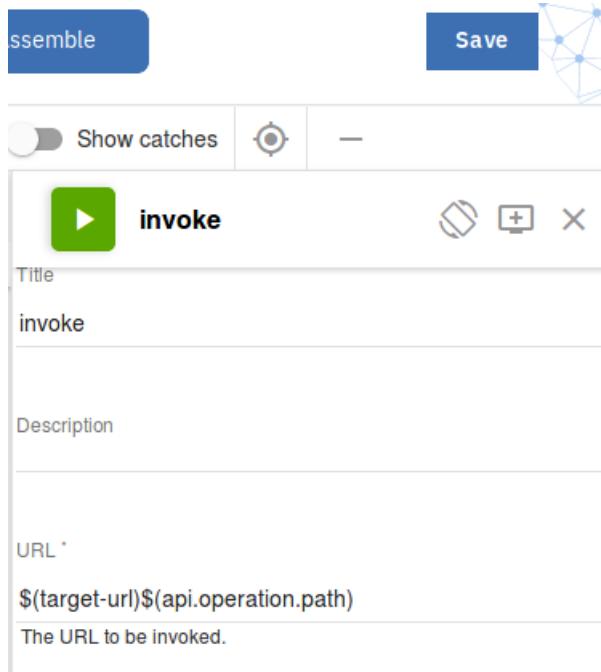
When the API Gateway receives a request for the `GET /estimate` API operation, it routes the request to the message processing policy. The **invoke** operation calls the implementation of the API, at the address that is specified by the URL.

In this case, the URL address is pre-filled with the variable named `$(target-url)`.

In some cases, you specify a URL address that is made up of variables that are defined in the API Manager, such as `$(target-url)${request.path}`.

The `target-url` is an API property that represents the host name of the API implementation. This value is different than the `host` variable, which represents the entry point for the API: the API Gateway.

- __ c. Modify the **URL** to `$(target-url)$(api.operation.path)`



Information

The **api.operation.path** property is '/estimate', while the **request.path** property is '/savings/estimate'. The **request.path** property includes the base path.

The **request.querystring** property is 'deposit=300&rate=0.04&years=20'. The query string does not include the question mark (?) character.

For more information, see the IBM Knowledge Center for API Connect and use the search string "API Connect context variables".

Note: In the exercise, the target-url is 'http://savingsample.mybluemix.net/api/Plans'.

The target-url was specified when you created the OpenAPI definition.

The **api.operation.path** / estimate is concatenated with the target-url.

The **request.querystring** parameters are then automatically added to the target.

- __ d. **Save** the changes.
- __ e. Close the properties view.

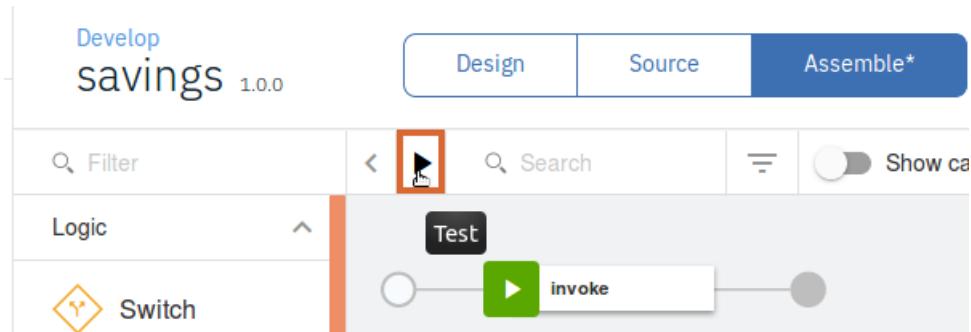
2.4. Test the API operation in the assembly

The API Manager includes a built-in test client for your API operations:

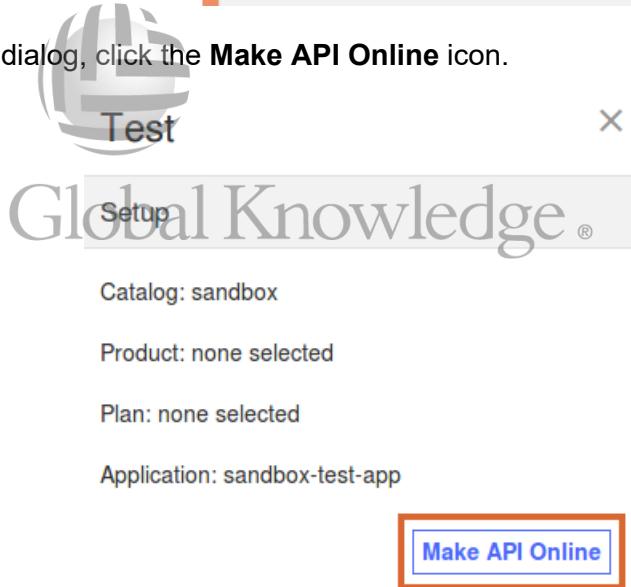
- In the **assemble** view, you can quickly invoke any operation in the API definition that you opened. For example, you can test the GET /estimate operation from the savings API.

In this section, test the API operation from the test client in the assemble view. When you confirmed that your API works correctly, you can make further changes to the API definition and policy assembly.

- 1. Test the get /estimate operation within the assemble view in the API Editor.
 - a. In the assemble view, click the **test** icon. The icon is between the filter and search fields.



- b. In the test dialog, click the **Make API Online** icon.



This option automatically publishes the API to the sandbox catalog and makes it callable on the gateway.

- ___ c. Select the `get /estimate` operation in the test client.

Test X

Setup

Catalog: sandbox

Product: savings-auto-product

Plan: Default Plan

Application: sandbox-test-app

[Republish product](#)

Operation

Choose an operation to invoke:

[Operation](#)

[get /estimate](#)

- ___ d. Type the following API operation parameters:

- deposit: 300
- rate: 0.04
- years: 20



Global Knowledge ®

- __ e. After the parameters, you can repeat the API call multiple times. Leave the repeat check box cleared and the stop on error box selected.

Years of saving
years *

20

Generate

Repeat

Repeat the API invocation a set number of times, or until the stop button is clicked

Stop after:
10

Stop on error

Invoke

Click **Invoke**.

- __ f. Examine the result from the API operation call.

Response

Status code: -1

No response received. Causes include a lack of CORS support on the target server, the server being unavailable, or an untrusted certificate being encountered.

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 again.

<https://apigw.think.ibm/think/sandbox/savings/estimate?deposit=300&rate=0.04&years=20>

Response time:
44ms



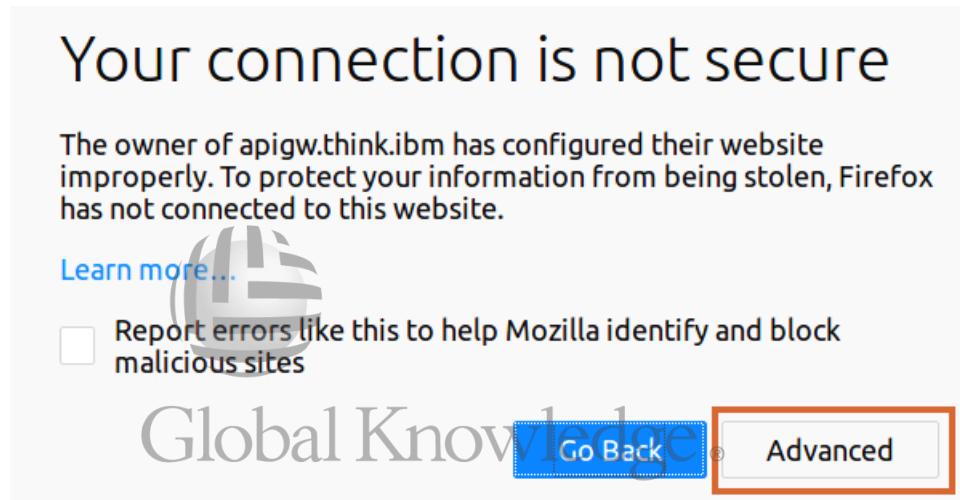
Information

Why did the test client API call fail?

The API Manager web application makes an HTTP request to the gateway. The gateway uses a self-signed security certificate for SSL/TLS traffic. Web browsers do not accept self-signed security certificates.

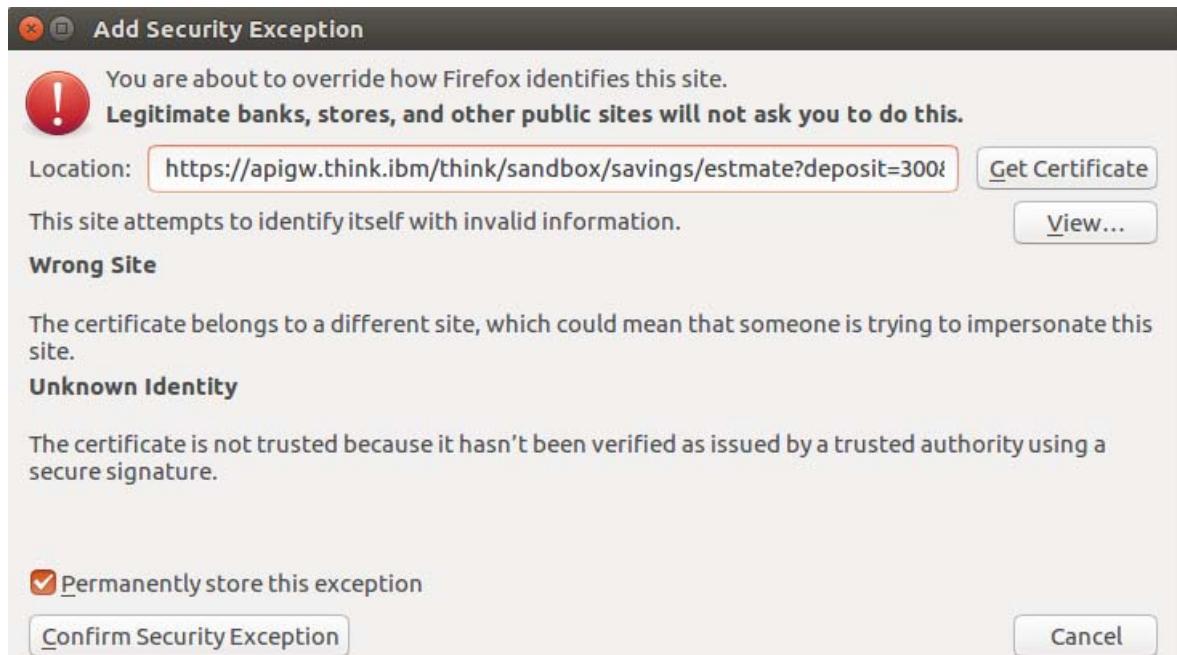
To fix this issue, you must create a security exception for the call to the gateway in your web browser. Accept the self-signed security certificate for the micro gateway with a security exception.

- ___ g. Click the link in the response area.
- ___ 2. Accept the security exception from the web browser.
 - ___ a. Click **Advanced**.



- ___ b. Click **Add exception**.

- __ c. Click **Confirm security exception**.



- __ d. You see a result in the browser.

- __ 3. Invoke the GET /estimate API operation at the gateway again.
__ a. In the API test client, click **Invoke**.

- ___ b. Confirm that the gateway returns the correct balance value in the responses section.

Response

Status code:

200 OK

Response time:

315ms

Headers:

apim-debug-trans-id: 426530149-Landlord-apiconnect-2269d2b5-d28a-431f-89cb-65556c19c6eb
content-type: application/json; charset=utf-8
x-global-transaction-id: 196c55655c463a3600000592
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,99;

Body:

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



Global Knowledge ®

2.5. Define a POST API operation with a JSON request message

The Savings Plan REST API sample application also provides a `POST /estimate` operation. The operation accepts the deposit, rate, and years input parameter as a JSON object in the HTTP request message:

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

You create a POST operation to your front-end savings API that implements a call similar to this curl command:

```
$ curl --header 'Content-Type: application/json' --request POST --data
'{ "deposit":300,"rate":0.04,"years":20}' --url
'http://savingsample.mybluemix.net/api/Plans/estimate'
```

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

In this section, create an API operation that is named `POST /estimate` in your savings OpenAPI definition. Define a schema object that is named `plan` with the `deposit`, `rate`, and `years` properties. Add an **invoke** policy to make a POST request to the API implementation at `savingsample.mybluemix.net`.

- ___ 1. Create a schema definition named `plan`.
 - ___ a. In the savings API definition, switch to the Design view.
 - ___ b. Select the **Definitions** section.
 - ___ c. Select **Add** to create a schema definition.
 - ___ d. Change the schema definition name to `plan`.

Edit definitions

Name

Type

Description (optional)

- ___ e. Change the description to `Parameters for a savings plan calculation`.

- ___ f. In the properties section, click Add three times. Then, add the deposit, rate, and years properties.

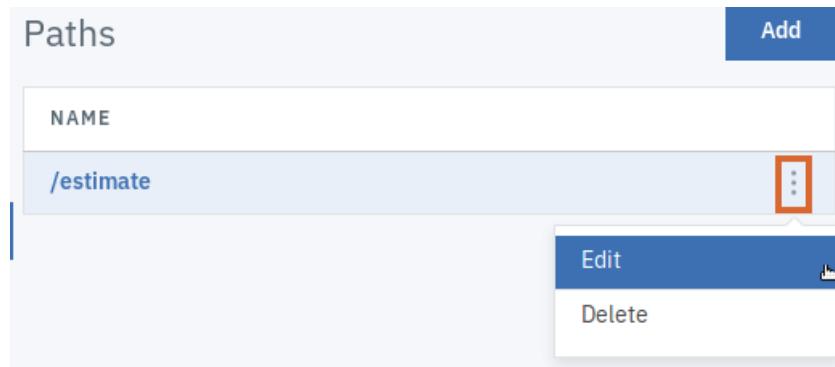
Properties

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

Table 3. Plan definition properties

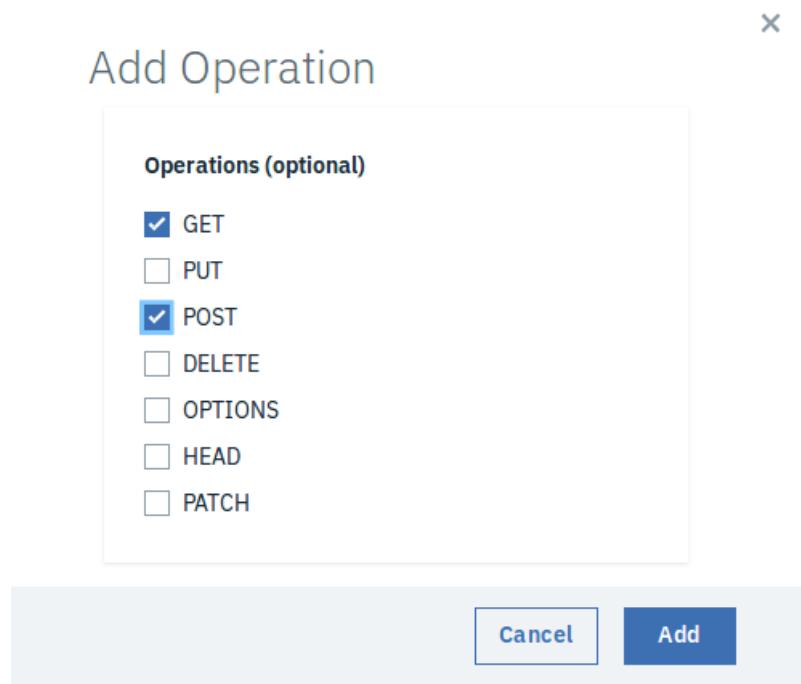
Property name	Description	Type	Example
deposit	Annual deposit	float	300.00
rate	Interest rate	float	0.04
years	Years	integer	20

- ___ g. Click **Save**.
- ___ 2. Create the POST /estimate API operation.
- ___ a. Select the **Paths** section.
- ___ b. Click the ellipsis alongside the /estimate path. Then, click **Edit**.



- ___ c. In the /estimate API path, click **Add** alongside Operations.

- ___ d. Select POST and leave GET selected.



Click **Add**.
The POST operation is added.

- ___ e. Click **Save**.
- ___ f. If necessary, open the paths again with the **Edit** option.
- ___ g. Click the ellipsis (three dots) alongside the POST operation. Then, click **Edit**.
- ___ h. In the `POST /estimate` operation, click **Add** alongside Parameters.
- ___ i. Add the `plan` object as the input parameter.

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

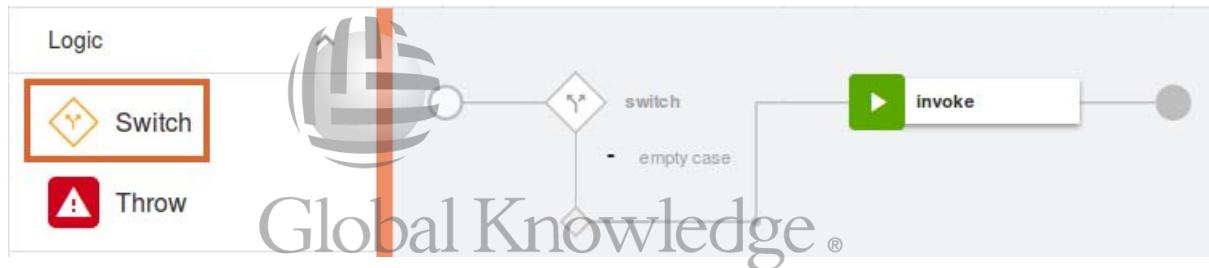
Table 4. `POST /Plans/estimate` input parameters

Name	Located in	Description	Required	Type
plan	Body	Savings plan estimate details	Yes	plan

- ___ j. Click **Add** alongside Response.
- ___ k. In the response area, type 200 in the status code, set the schema to plan, and description to 200 OK.

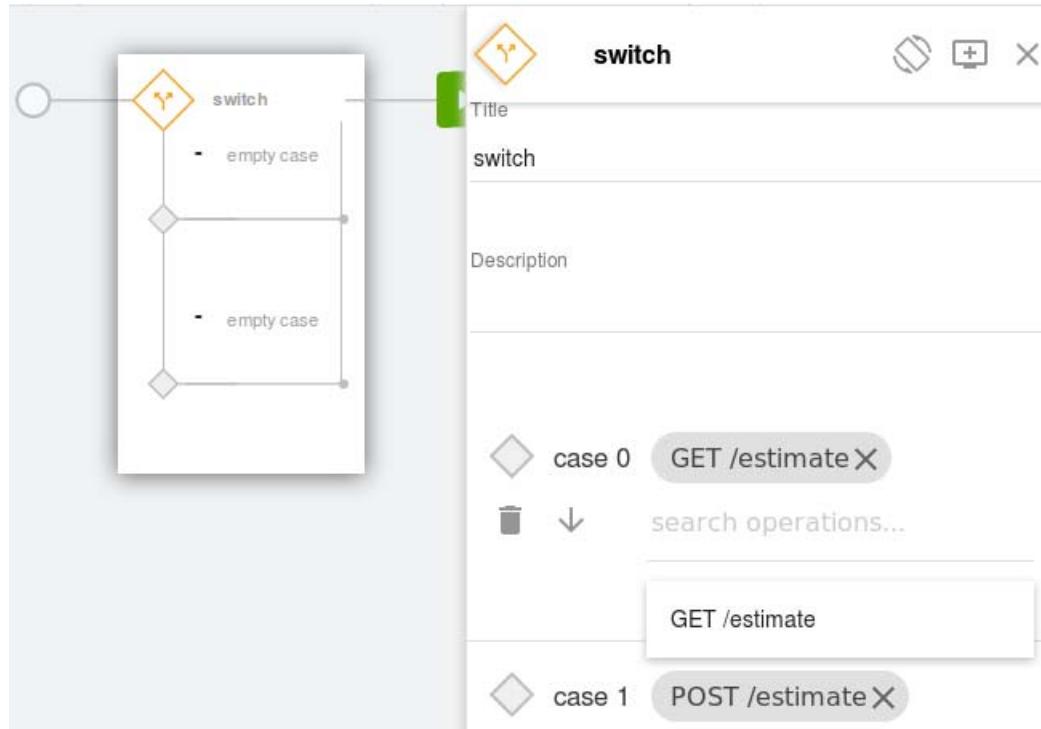
Parameters		REQUIRED	NAME	LOCATED IN	TYPE
<input checked="" type="checkbox"/>	plan	body	plan		

- ___ 3. **Save** the changes.
- ___ 4. Create a switch policy to handle two types of API operation request: GET /estimate and POST /estimate.
 - ___ a. Select the **Assemble** view.
 - ___ b. Add a switch policy at the beginning of the message processing pipeline by selecting the Switch from the palette, then drop it between the start and invoke icons on the free-form area.



- ___ c. In the properties editor for the switch policy, set the case 0 branch to GET /estimate.
- ___ d. Select **+ Case**.

- e. Set the case 1 branch to POST /estimate.



- f. Save and close the operation-switch properties editor.
5. Add the **invoke** policy to the GET /estimate case.
- a. Select the existing **invoke** policy in the assembly palette.
- b. Drag the existing **invoke** policy over the GET /estimate case.



- ___ c. Drop the existing **invoke** policy in the highlighted section in the GET /estimate case.



- ___ 6. Add an **invoke** policy in the POST /estimate case.

- From the policy palette, select the **invoke** policy.
- Drag the **invoke** policy icon into the POST /estimate case. This case is the second entry in the switch construct.
- In the properties editor, change the title to **invoke_post**.
- Set the URL to `$(target-url)$(api.operation.path)`



- Scroll down to the HTTP Method.
- Set the invoke method to POST.
- Scroll down the properties page.
- Clear the **Stop on error** check box.

- ___ 7. **Save** the changes.

2.6. Test the savings API with the assembly test feature

In this section, test the POST operation from the built-in test client.

- ___ 1. Test the POST /estimate API operation.
 - ___ a. In the assembly view, click the Test icon.
 - ___ b. Select **Republish product** in the Test dialog.

Test X

Setup

Catalog: sandbox

Product: savings-auto-product

Plan: Default Plan

Application: sandbox-test-app

 Republish product

The product is republished.

Operation
post /estimate

parameters

```
plan *
{
  "deposit": "300.00",
  "rate": "0.04",
  "years": "20"
}
```

[Show schema](#) | [Generate](#)

The sample values that you typed when setting the plan definition are inserted in JSON format. Remove the quotation marks that surround the numeric values.

- __ e. Click **Invoke**.
- __ f. Confirm that the API implementation returns a status of 200 OK, with a balance value of 8933.42.

Response

Status code:
200 OK

Response time:
261ms

Headers:

```
apim-debug-trans-id: 426530149-Landlord-
apiconnect-e7ac5d43-b21d-4513-
bb29-65556c19a20e
content-type: application/json; charset=utf-8
x-global-transaction-
id: 196c55655c466529000006f2
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,93;
```

Body:

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

**Note**

If you get a 404 Not Found error, verify that you typed the target-url value correctly. Retype the value. Republish the product. Then, click **Invoke**.

End of exercise

Exercise review and wrap-up

In the first part of the exercise, you created an OpenAPI definition in the API Manager web application from an existing API. You defined the API paths, operations, request, and response messages that describe the saving sample REST API application. You also reviewed a copy of the API application that is hosted on IBM Cloud.

In the second part of the exercise, you defined a second API operation for the saving sample REST API. The first example used HTTP query parameters, and the second example embeds input parameters as a JSON object in the message body. You specified the data type schema as an OpenAPI schema type definition.

You tested the API definitions in the assembly view test feature of API Manager.



Exercise 3. Define an API that calls an existing SOAP service

Estimated time

00:30

Overview

With API Connect, you can define an API from existing enterprise services. In this exercise, you define an API that calls an existing SOAP service. You use the API Manager feature to create an API definition from an existing WSDL service. The imported WSDL defines the API paths and methods that map to SOAP web service operations, and map SOAP message types to API data types. You test the SOAP API in the test feature of API Manager.

Objectives

After completing this exercise, you should be able to:

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

Introduction

For existing API implementations, you can expose the API operations at the API Gateway. You create an API definition: a document that specifies a list of API paths, methods, expected request, and response messages.

In this scenario, your organization already developed a set of SOAP web services. SOAP is a mature, remote service standard that is common with large enterprise architectures. The goal of this exercise is to take the operations from the existing SOAP service and make it available at the API Gateway.

In effect, you build a SOAP web service proxy for an existing service. This configuration does not convert a SOAP requests to an alternative format. The concept of a SOAP-to-REST API bridge is another topic that is not covered in this exercise.

Requirements

You must complete this lab exercise in the student development workstation: the Ubuntu host environment. This virtual machine is preconfigured with the API Connect components so that you can create, edit, and test APIs.



Exercise instructions

Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



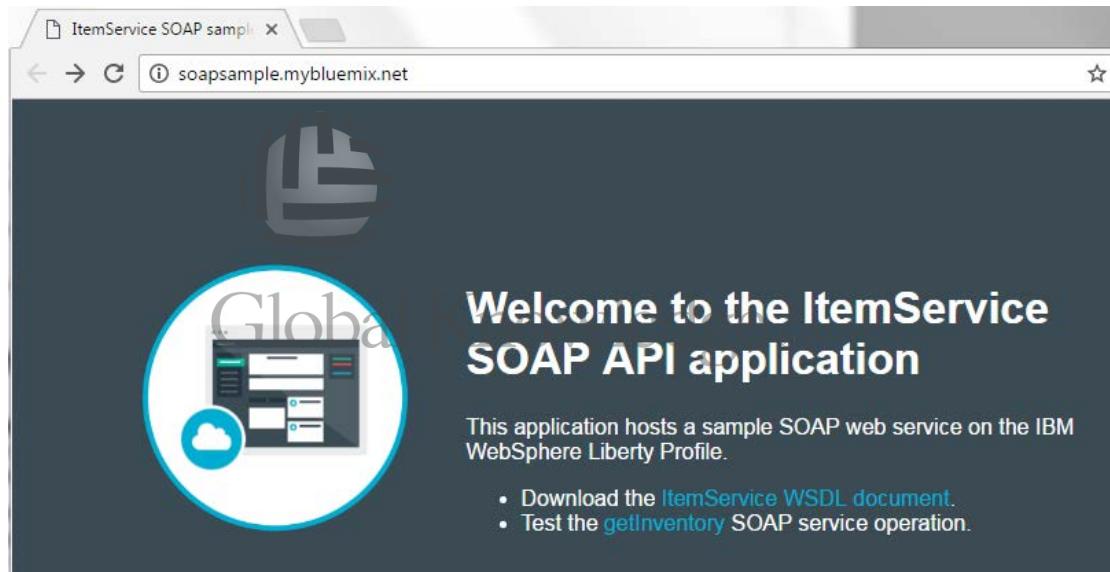
3.1. Review the existing SOAP web service

The `ItemService` application maintains a list of vintage IBM products from its corporate history. The application hosts a remote service that returns a list of items in the store inventory. Unlike many REST services on API Connect, the `ItemService` remote service is built as a SOAP web service.

SOAP is an application protocol for remote service invocation over HTTP connections. SOAP services store the operation command and data in a custom XML data format known as an SOAP envelope. This design is in contrast to modern REST APIs, which uses HTTP methods as the command and store application data in the HTTP message.

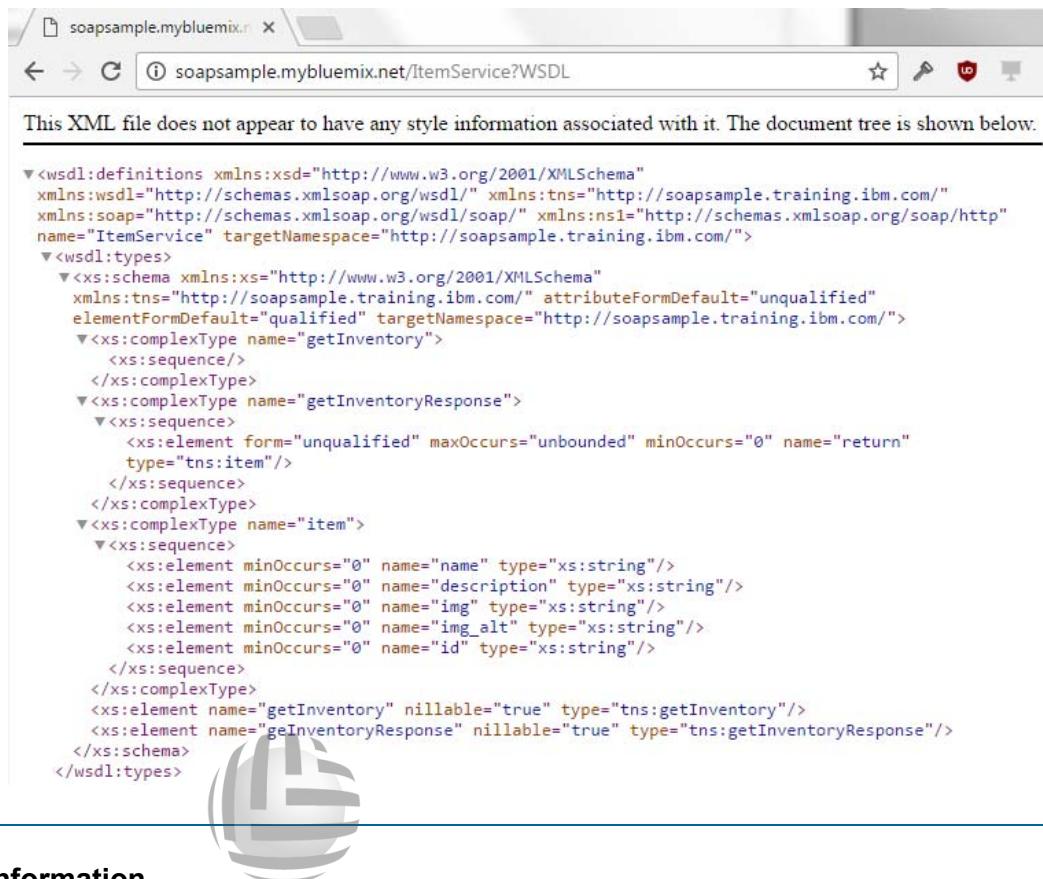
In this section, you test and verify the `ItemService` SOAP service. You retrieve and review a copy of the SOAP service interface, in the form of a Web Services Description Language (WSDL) document.

- 1. Open the `ItemService` website.
 - a. Open the <http://soapsample.mybluemix.net> page in a web browser tab on the student image.



- b. Click the **ItemService WSDL document** link on the main page.

- ___ c. Examine the contents of the **ItemService** WSDL document.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<wsdl:definitions xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://soapsample.training.ibm.com/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
    name="ItemService" targetNamespace="http://soapsample.training.ibm.com/">
  <wsdl:types>
    <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://soapsample.training.ibm.com/" attributeFormDefault="unqualified"
      elementFormDefault="qualified" targetNamespace="http://soapsample.training.ibm.com/">
      <xss:complexType name="getInventory">
        <xss:sequence/>
      </xss:complexType>
      <xss:complexType name="getInventoryResponse">
        <xss:sequence>
          <xss:element form="unqualified" maxOccurs="unbounded" minOccurs="0" name="return"
            type="tns:item"/>
        </xss:sequence>
      </xss:complexType>
      <xss:complexType name="item">
        <xss:sequence>
          <xss:element minOccurs="0" name="name" type="xs:string"/>
          <xss:element minOccurs="0" name="description" type="xs:string"/>
          <xss:element minOccurs="0" name="img" type="xs:string"/>
          <xss:element minOccurs="0" name="img_alt" type="xs:string"/>
          <xss:element minOccurs="0" name="id" type="xs:string"/>
        </xss:sequence>
      </xss:complexType>
      <xss:element name="getInventory" nillable="true" type="tns:getInventory"/>
      <xss:element name="getInventoryResponse" nillable="true" type="tns:getInventoryResponse"/>
    </xsschema>
  </wsdl:types>

```



Information



What is the purpose of the Web Services Description Language (WSDL) document?

The purpose of the WSDL document is two-fold: to describe the service interface, and to specify the network endpoint and protocol bindings for the web service.

The service interface describes all the details that a client application requires to call the web service. In this document, the schema section lists two XML data structures: the request and response message for the service operations. The `item` complex type describes the structure of the item model object: five fields that describe the details of an item in the store inventory.

The portType section lists the names of the operations in the web service. In this example, ItemService has one operation, named `getInventory`. The expected request message is defined in an XML element named `getInventory`. The response message is an XML element named `getInventoryResponse`.

The rest of the document describes how to connect and call the SOAP service over the network. The service section lists the service endpoint as

`http://soapsample.mybluemix.net/ItemService`. The bindings section explains how to construct an HTTP request message for the service, and how to interpret the HTTP response message from the same service.

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

- __ 2. Test the getInventory SOAP service operation.
 - __ a. Return to the ItemService main page.
 - __ b. Click Test the **getInventory** SOAP service operation.
 - __ c. Review the SOAP request and response messages.

The screenshot shows a web browser window with the URL soapsample.mybluemix.net/getInventory.html. The page title is "ItemService SOAP API Sample". A sub-header states, "This test client invokes the getInventory operation from the SOAP service." Below this, a section titled "SOAP request:" contains the following XML code:

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

Below the request, a section titled "SOAP response:" shows the HTTP status code 200. The response body contains a large block of XML text describing historical IBM equipment, specifically mentioning the Dayton Meat Chopper and the Hollerith Tabulator.



Information

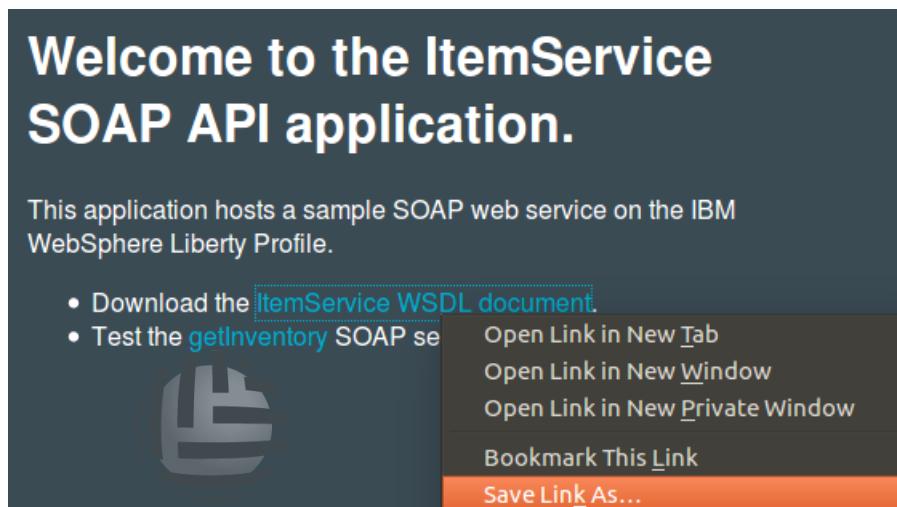
The ItemService website includes a test client for the SOAP service. This page sends an SOAP request message to the `http://soapsample.mybluemix.net/ItemService` endpoint.

The first half of the page displays the SOAP request message. The HTTP request message stores an XML document in the SOAP envelope format. In the SOAP message body, the `<tns:getInventory />` XML element represents the name of the SOAP operation.

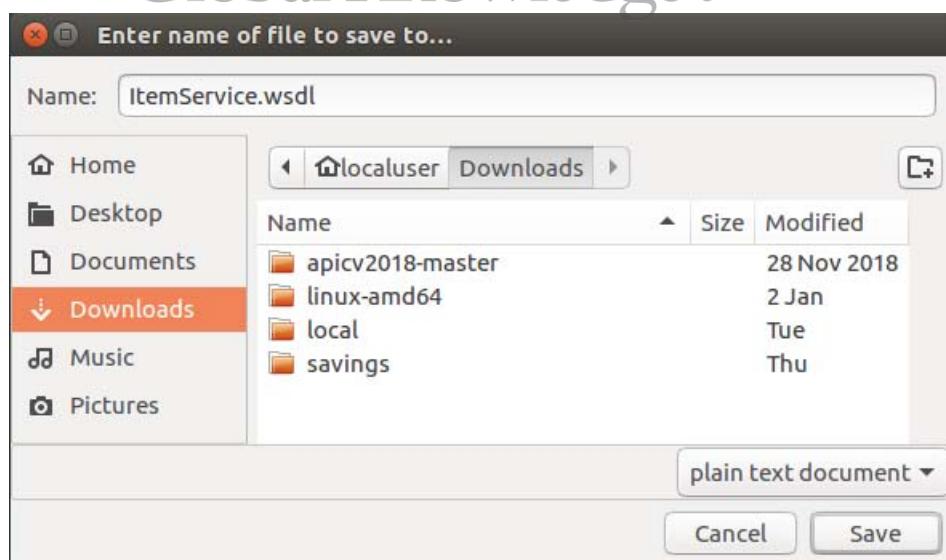
The second half of the page displays the response from the SOAP service. The HTTP status code of 200 indicates that the SOAP service processed the request successfully. The HTTP response message stores another XML document in the SOAP envelope format.

In this example, the getInventory service returned the names and descriptions of items in the inventory.

- ___ 3. Download a copy of the ItemService WSDL document.
 - ___ a. Return to the ItemService main page.
 - ___ b. Right-click the **ItemService WSDL document** link.
 - ___ c. Click **Save link as**.



- ___ d. Rename the document to `ItemService.wsdl`.



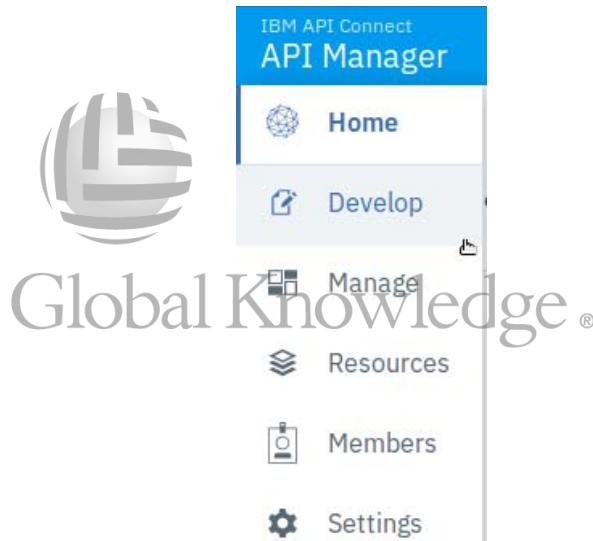
- ___ e. Click **Save**.
By default, the file is saved to the `/home/localuser/Downloads` directory.
- ___ f. Close the web browser with the sample SOAP application.

3.2. Create a SOAP API definition from a WSDL document

In this section, you generate an OpenAPI 2.0 API definition from an existing SOAP service. Import the WSDL document into the API Manager. Examine the API definition and message processing policies for the API.

- 1. Sign in to API Manager.
 - a. Open a browser window.
Then, type `https://manager.think.ibm.`
 - b. Type the credentials:
 - User name: ThinkOwner
 - Password: Passw0rd!
- Click **Sign in**.
You are signed in to API Manager.

- 2. Select the Develop option from the side menu.



- 3. Create an API definition that is named `ItemService` from the `ItemService` WSDL document.
 - a. In the API Manager develop page, click **ADD**.
Then, select **API**.

- __ b. On the Add API page, select **From existing WSDL service (SOAP proxy)**.

Add API

Create

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

The selector is highlighted.

- __ c. Click **Next**.
__ d. In the Create from page, click **Browse**.

Create from

Select the target wsdl file to create from

Global Knowledge®
Drag & Drop
your file here, or

Browse

Cancel **Next**

- __ e. Select the `ItemService.wsdl` document from the directory that you saved it in an earlier step.
__ f. The WSDL is imported and validated.
__ g. Click **Next**.

- __ h. The ItemService from the imported WSDL is selected.

Select a WSDL service from the imported file

TITLE	DESCRIPTION
<input checked="" type="checkbox"/> ItemService	getInventory

Back **Cancel** **Next**



Information

The new API from WSDL parsed through the WSDL document and found one SOAP service named `ItemService`. The service defines one operation, named `getInventory`.

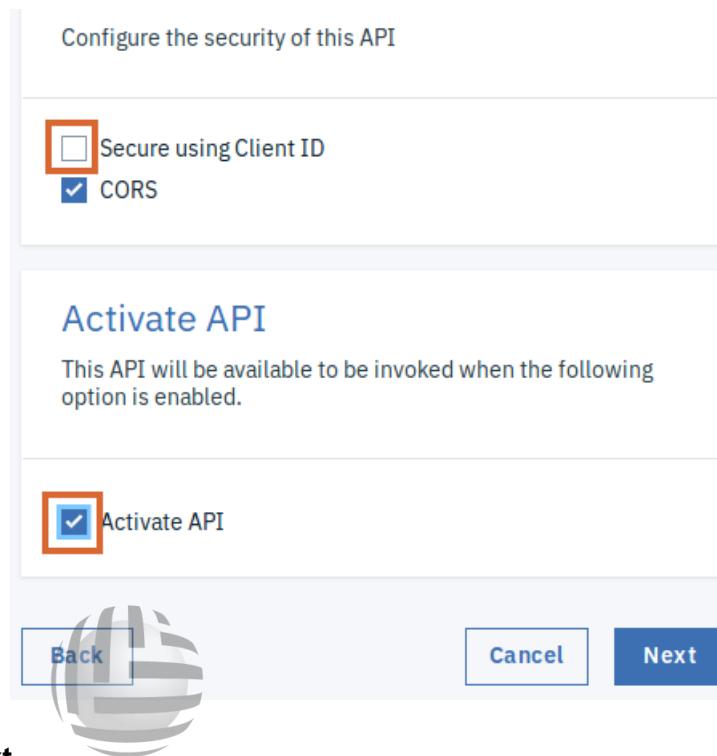
- __ i. Click **Next**.
The ItemService API definition is created.
__ 4. Click **Next**.



Global Knowledge ®

5. On the configure the security of this API page:

- Secure using Client ID: cleared
- CORS: selected
- Activate API: selected



Click **Next**.



Information

Global Knowledge®

By default, all API definitions that you create in the API Manager includes the API Key security requirement. This requirement forces every API caller to supply a valid `client_ID` value. That is, application developers must register their application in the Developer Portal.

To allow unregistered applications access to the API, clear the Client ID security requirement in the API definition.

NOTE: When you select the option Activate API, API Manager automatically creates a product and an API subscription and generates a client ID and client secret.

- ___ 6. The Summary window is displayed.

The screenshot shows a summary window for an API. At the top, there are two green checkmarks: one indicating a "Generated OpenAPI 2.0 definition" and another indicating that "Your API is online!". Below this, the "API Base URL" is listed as `https://apigw.think.ibm/think/sandbox/ItemService`, with a copy icon to its right. Further down, under "API Subscription", there are fields for "Client ID" containing the value `41115dff8a42e59841adbc774dbefc87` and "Client Secret" containing a long encoded string `OKBfzFMW31Vz6getBWStWydUYqjucsMNkYHnvYRe1eo=`, each with a copy icon to its right. A large watermark for "Global Knowledge" is overlaid across the bottom of the window.

- ___ 7. Review the API definition

- ___ a. Click **Edit API**.

- ___ b. The API definition is displayed in the Design view.

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

- Develop** tab is selected.
- ItemService** 1.0.0 is the current API.
- Design** tab is selected.
- API Setup** sidebar is open, showing options: Security Definitions, Security, Paths, Definitions, Properties, Target Services, Categories, and Activity Log.
- Info** section:
 - Summary: Enter the API summary details.
 - Title**: ItemService
 - Name**: itemservice
 - Version**: 1.0.0

- ___ c. Scroll down with the API setup selected and review the base path.

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

- Develop** tab is selected.
- ItemService** 1.0.0 is the current API.
- Design** tab is selected.
- API Setup** sidebar is open, showing options: Security Definitions, Security, Paths, Definitions, Properties, and Target Services.
- Global Knowledge Base Path** section:
 - Description: The base path is the initial URL segment of the API and does not include the host name or any additional segments for paths or operations.
 - Base path (optional)**: /ItemService



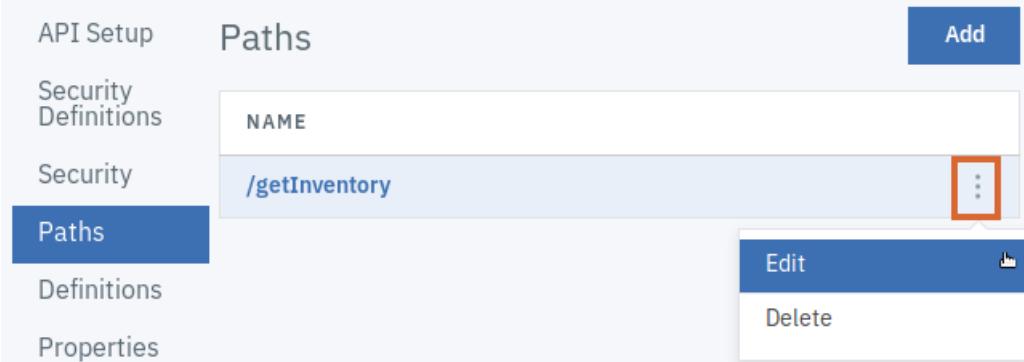
Information

The base path section describes the network endpoint on the API Gateway for requests to the ItemService SOAP API.

Examine the consumes and produces section of the API definition. The ItemService API expects HTTP requests with a `text/xml` media type. It returns HTTP responses with an `application/xml` media type.

In effect, the itemservice API receives and sends SOAP messages. This message type is in contrast to REST APIs, which use JavaScript Object Notation (JSON) as the data type.

-
- ___ 8. Review the message processing policies in the Itemservice API definition.
 - ___ a. Select the **Paths** section.
 - ___ b. A single path that is named `/getInventory` is displayed. Click the ellipsis in the paths area. Then, click **Edit**.



The screenshot shows the 'Paths' tab selected in the API Setup interface. A path named '/getInventory' is listed in the NAME column. A context menu is open over this entry, with 'Edit' highlighted. Other options in the menu include 'Delete' and a small icon.

- ___ c. The Itemservice API defines one POST API operation. Click the option to edit the POST operation.



In the SOAP request message, this operation expects an input message in the body of the message named `getInventoryInput`. It returns a SOAP response message with an output message of `getInventoryOutput`.

-
- ___ d. Click Cancel when you are finished reviewing the POST operation.
 - ___ 9. Review the API definitions.
 - ___ a. Return to the ItemService definition main page in the Design view.
 - ___ b. Click **Definitions**.
A number of schema definition objects are already defined.
 - ___ 10. Examine the message processing policies for the Itemservice API definition.
 - ___ a. Click the **Assemble** tab.

- __ b. Select the **invoke** policy in the assembly flow.

The screenshot shows the Assemble view of the API Connect interface. At the top, there are tabs for Design, Source, and Assemble*, with Assemble* being the active tab. To the right of the tabs is a Save button. Below the tabs, there are search and filter tools, and a toggle for 'Show catches'. The main area displays a flow diagram with a single node labeled 'invoke' highlighted by a red box. To the right of the node, there is a configuration panel for the 'invoke' action. The configuration fields are as follows:

- Title: invoke
- Description: (empty)
- URL *: <https://soapsample.mybluemix.net/ItemService>
The URL to be invoked.



Information

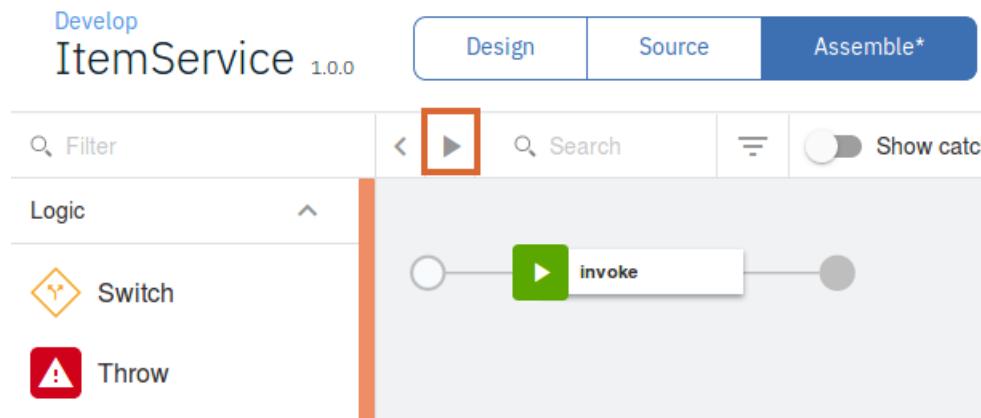
The assemble view displays the message processing policies for all operations in the Itemservice API. At run time, the API Gateway enforces these policies on every request message that clients send to the API.

The Itemservice API definition includes one policy: an invoke action. The purpose of this policy is to forward the HTTP request message to the SOAP service implementation at <http://soapsample.mybluemix.net/ItemService>.

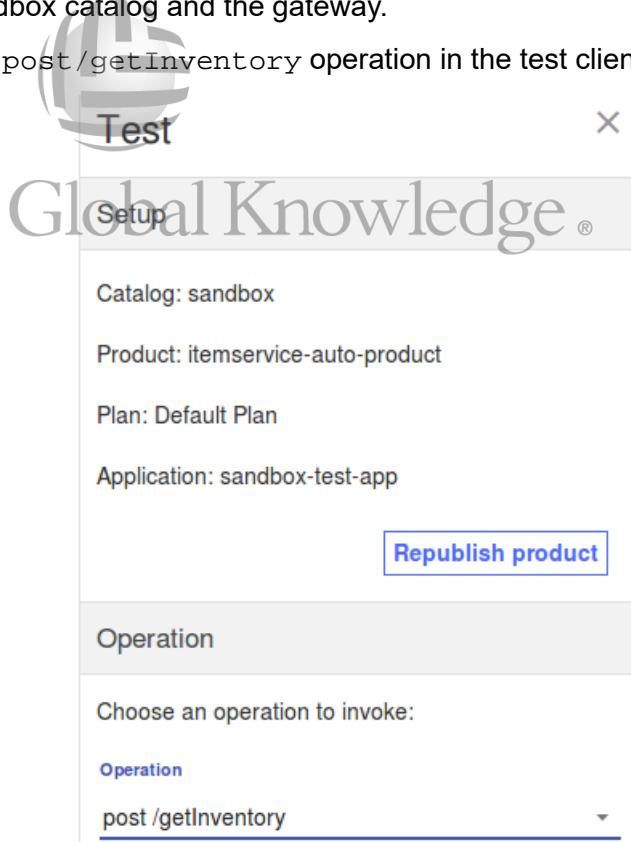
3.3. Test the API on the DataPower gateway

In this part, you test an API that contains DataPower policies on the DataPower gateway. You use the test feature of the assembly to test the API.

- ___ 11. Test the Itemservice API from the test option in the assembly of the API Manager.
 - ___ a. In the Assembly editor, click the Test icon.



- ___ b. Since you selected the Make API active option earlier, the product is already published to the sandbox catalog and the gateway.
- ___ c. Select the `post /getInventory` operation in the test client.



- ___ d. A client ID is automatically inserted into the test client.
- ___ e. Scroll down to the body area in the test client.

- ___ f. Click in the body area of the test client. Then, click **Generate** to create some data in the body of the message.

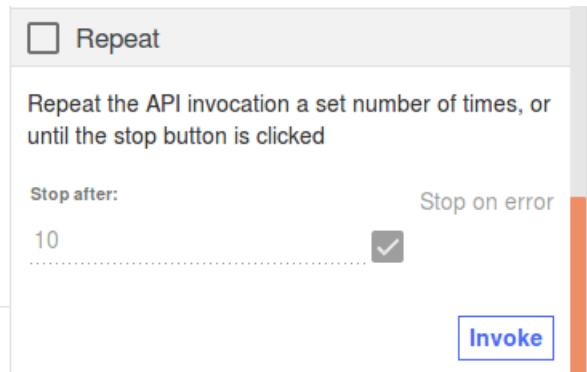


- ___ g. The body area is populated with data.

The screenshot shows a test client window titled 'Test'. Inside, there's a 'parameters' section with a 'body *' input field. The field contains the following XML code:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org
  /soap/envelope/">
  <soapenv:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004
      /01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004
      /01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>string</wsse:Username>
        <wsse:Password>string</wsse:Password>
        <wsse:Nonce>
```

- ___ h. Scroll to the bottom of the test client.
Then, click **Invoke**.



- ___ i. The result is displayed in the test window.

Response

Status code:
200 OK

Response time:
491ms

Headers:

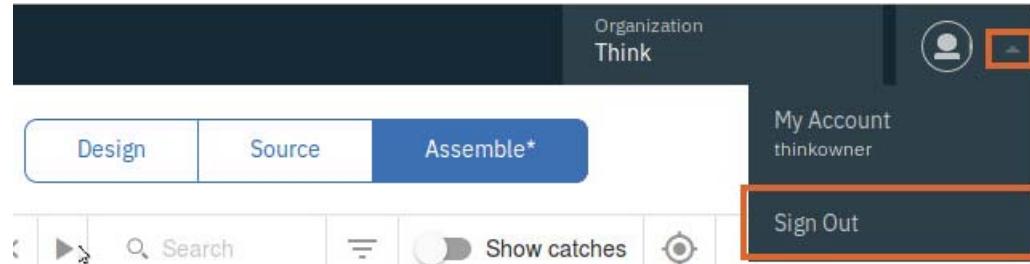
apim-debug-trans-id: 426530149-Landlord-
apiconnect-7f16679a-8f84-4ab4-93ca-
65556c191c32
content-language: en-US
content-type: text/xml; charset=UTF-8
x-global-transaction-
id: 196c55655c80080d000005c2
x-ratelimit-limit: name=rate-limit,100;
x-ratelimit-remaining: name=rate-limit,99;

Body:

```
<soap:Envelope  
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  
    <soap:Body>  
        <ns2:geInventoryResponse>
```

- ___ j. Close the test window.

__ 12. Sign out of API Manager.



__ 13. Close the browser.

End of exercise



Exercise review and wrap-up

In the exercise, you created an OpenAPI definition for an existing SOAP service. Specifically, you generated a SOAP API definition based on an existing SOAP service interface: the WSDL document.

You tested the SOAP API definition in the API Manager assembly test feature.



Exercise 4. Create a LoopBack application

Estimated time

00:45

Overview

In this exercise, you build a LoopBack application to implement an API. You generate the application scaffold with the apic command-line utility. You examine the generated files, the model, and properties of the Loopback application. You run the generated Node application and test the operations with the Loopback API Explorer.

Objectives

After completing this exercise, you should be able to:

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

Introduction

Global Knowledge®

The LoopBack framework is a framework to build REST APIs in the Node.js programming language. This open source framework provides nearly codeless API composition, isomorphic models, and a set of Node.js modules that you can use independently or together to quickly build applications that expose REST APIs.

An application interacts with data sources through the LoopBack model API, available locally within Node.js or remotely over REST. Using these APIs, apps can query databases, store data, upload files, send emails, create push notifications, register users, and perform other actions that are provided by data sources and services.

In this exercise, you examine the sample API implementation for the note application: an API to create, retrieve, update, and delete a set of text notes. Identify the structure of a LoopBack application. Define and build model objects with the command line utility. Review and test the REST APIs with the Loopback API explorer.

Requirements

You must complete this lab exercise on the student remote image: the Ubuntu host environment. This virtual machine is pre-configured with the Node runtime environment, the 'npm' Node package manager, and the IBM API Connect toolkit.

Exercise instructions

Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



Global Knowledge®

4.1. Create a LoopBack application with the apic command line utility

The LoopBack application is a Node.js application that implements a REST API based on a set of models. Generate a sample LoopBack application with the apic command line utility.

- ___ 1. Create a directory to hold the contents of the LoopBack application.
 - ___ a. Open a terminal window from the Ubuntu desktop launcher.
 - ___ b. Create two directories, samples and samples/notes.

```
$ cd ~
$ mkdir samples samples/notes
$ cd samples/notes
```

- ___ c. Verify that your current working directory is the sample/notes directory.

```
$ pwd
/home/localuser/samples/notes
```

- ___ 2. Generate the Loopback 'notes' sample application.

- ___ a. Run the LoopBack application generator in the 'apic' command line utility.

```
$ apic lb
```

- ___ b. Select 'notes' as the application name.

- ___ c. Select the 'notes (A project containing a basic working example, including a memory database)' project as the starting point for the application.

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

? What kind of application do you have in mind? (Use arrow keys)

empty-server (An empty LoopBack API, without any configured models or data sources)

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

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

- ___ d. Wait until the generator finishes creating the application.

4.2. Examine the structure of a LoopBack application

Before you test the sample 'notes' LoopBack application, explore the structure of the application scaffolding. Review the API definition files, the configuration settings, the model properties, and the model source code.

- ___ 1. Review the structure of the application.
 - ___ a. List the contents of the root directory in the 'notes' application.

```
$ ls -F
client/
common/
node_modules/
package.json
package-lock.json
server/
```



Information

The application generator created the directory structure for a LoopBack application:

- The **client** directory stores the readme file and any client files the application uses.
- The **common** directory stores resources that the client and server application uses.
- The **server** directory stores the configuration settings, boot scripts, and data source settings for the server application.

Every LoopBack application is a Node.js application. Therefore, the notes application includes a node_modules directory to store local Node packages. The package.json application manifest stores metadata on the application, including name, version, package dependencies, and software licensing information.

- ___ b. Examine the contents of the **common** directory.

```
$ ls -F common
models/
$ ls -F common/models/
note.js  note.json
```



Information

At the core of a LoopBack application are the models: JavaScript configuration and code that represent the data models in the API. The LoopBack framework creates a set of data-centric create, retrieve, update, and delete operations for each model. In a later exercise, you explore how LoopBack connectors persist model data through data sources.

- ___ c. Examine the contents of the **server** directory.

```
$ ls -F server
boot/           datasources.json      model-config.json
component-config.json middleware.development.json server.js
config.json     middleware.json
```



Information

The **server** directory contains a set of configuration files that control the behavior of the LoopBack framework.

- The **datasources.json** file defines a list of LoopBack data sources. A data source is an object that provides programmatic access to a remote data source: a database, a remote service, or email server.
- The **middleware.json** file defines configuration settings for Express middleware modules. Express is a Node.js framework for web applications. LoopBack is built upon the Express framework.
- The **middleware.development.json** file defines environment properties for Express middleware modules.
- The **server.js** script is the main entry point to the LoopBack application. This script loads the LoopBack framework with the configuration files.
- The **config.json** file sets general parameters for the LoopBack application, such as the base path for the REST APIs.
- The **model-config.json** file maps each LoopBack module to a specific data source. The application stores the models in the common/models directory.

In a later exercise, you install and configure data sources with server configuration files. You also map LoopBack models that you create to data sources.

- ___ d. Examine the contents of the **server/boot** directory.

```
$ ls -F server/boot
authentication.js
root.js
$ more server/boot/root.js
'use strict';

module.exports = function(server) {
  // Install a '/' route that returns server status
  var router = server.loopback.Router();
  router.get('/', server.loopback.status());
  server.use(router);
};
```



Information

The **boot scripts** directory contains Node scripts that customize the LoopBack application behavior. In this example, the **root.js** script defines a default route that returns the LoopBack object status. In practice, you open this route at run time to confirm that the LoopBack application is running.

- 2. Review the LoopBack **note** model configuration and source code.

- a. Examine the contents of the **common/models** directory.

```
$ ls common/models
note.js  note.json
```

- b. Review the contents of the **note.json** model property file.

```
$ more common/models/note.json
{
  "name": "Note",
  "properties": {
    "title": {
      "type": "string",
      "required": true
    },
    "content": {
      "type": "string"
    }
  }
}
```



Global Knowledge®

- c. Review the contents of the **note.js** model script file.

```
$ more common/models/note.js
module.exports = function(note) {
};
```



Information

Two files define the properties and behavior of a LoopBack model:

- The **model configuration file** declares the name, properties, and relationships in the model object. You create a configuration file in the JavaScript object notation (JSON) format.
- The **model script file** defines any additional behavior beyond the standard create, retrieve, update, and delete operations that the LoopBack framework provides for every model object. You develop a model script file as a Node.js JavaScript anonymous function.

In this application, the **note.json** model configuration defines 'Note' as the name of the model object. The 'Note' model defines two properties: **title**, and **content**. Both properties store information in the JSON string format. The **title** is a required property.

The **note.js** model script file defines an empty anonymous function. By default, every LoopBack model object inherits its behavior from the model base class. Therefore, you do not need to implement any custom code to create data-centric REST API operations. You can implement remote methods or override the default operations with custom code in this script file.

___ 3. Review the in-memory database data source.

___ a. Examine the in-memory data source that is defined in the **server/datasources.json** configuration file.

```
$ more server/datasources.json
{
  "db": {
    "name": "db",
    "connector": "memory"
  }
}
```



Information

The **datasources.json** configuration file declares the name of each data source in the LoopBack application. Recall that a data source is a JavaScript object that represents an external data store. Examples include relational databases, non-relational data stores, and remote services.

In this example, the LoopBack application creates a data source that is named 'db'. The data source maps to the "memory" LoopBack connector: an in-memory data store that persists model object data while the application is running for testing purposes.

Global Knowledge®

- __ b. Examine the model mapping to data sources in the **server/model-config.json** configuration file.

```
$ more server/model-config.json
{
  "_meta": {
    "sources": [
      "loopback/common/models",
      "loopback/server/models",
      "../common/models",
      "./models"
    ],
    "mixins": [
      "loopback/common/mixins",
      "loopback/server/mixins",
      "../common/mixins",
      "./mixins"
    ]
  },
  "User": {
    "dataSource": "db"
  },
  "AccessToken": {
    "dataSource": "db",
    "public": false
  },
  "ACL": {
    "dataSource": "db",
    "public": false
  },
  "RoleMapping": {
    "dataSource": "db",
    "public": false,
    "options": {
      "strictObjectIDCoercion": true
    }
  },
  "Role": {
    "dataSource": "db",
    "public": false
  },
  "Note": {
    "dataSource": "db"
  }
}
```



Information

The **model-config.json** configuration file maps LoopBack models in your application to a data source. In this example, the application mapped the 'Note' model object to the 'db' data source. Recall that the `datasources.json` configuration file created an in-memory database that is named 'db'.

You can map each model to one data source. If you do not want to persist model data, or link a model to a data source, leave out a model mapping in the `model-config.json` file.

The '`_meta`' and '`mixins`' section of the configuration file are preset values that the LoopBack framework code expects. Leave this section unchanged.

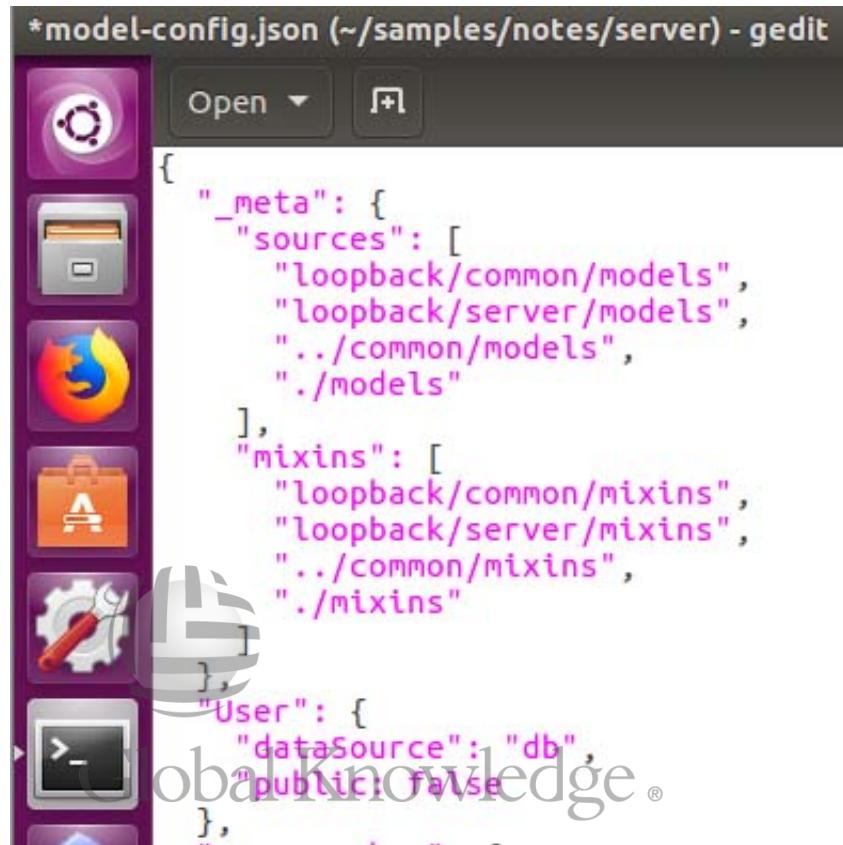
-
- ___ 4. Hide the API operations for the **user** model.
 - ___ a. Open the **model-config.json** file with the gedit application.
 \$ `gedit server/model-config.json`
 - ___ b. Add a property named `public` to the `user` object.



Global Knowledge®

- c. Set the value of the **public** property to **false**.

```
"User": {
  "dataSource": "db",
  "public": false
}
```



```
{
  "_meta": {
    "sources": [
      "loopback/common/models",
      "loopback/server/models",
      "../common/models",
      "./models"
    ],
    "mixins": [
      "loopback/common/mixins",
      "loopback/server/mixins",
      "../common/mixins",
      "./mixins"
    ],
    "User": {
      "dataSource": "db",
      "public": false
    }
  }
}
```

- d. Save the changes to the `model-config.json` file.
 e. Close the gedit application.



Questions

Why did you hide the API operations for the user model?

The user model is a built-in object in the LoopBack framework. With the model, you can configure role-based access control to your API. For this exercise, you configure notes as a publicly accessible service with no security restrictions. Therefore, you can safely hide the user API operations.

For more information about the User REST API, see the LoopBack documentation
<http://loopback.io/doc/en/lb3/User-REST-API.html>.

- ___ 5. Generate the API definition file for the application.

- ___ a. Ensure that you are in the notes directory.

```
$ pwd  
/home/localuser/samples/notes
```

- ___ b. In the terminal, type the commands to create the API definition file.

```
$ apic lb export-api-def > notes.yaml
```

The OpenAPI definition that is named `notes.yaml` is created in the notes directory.

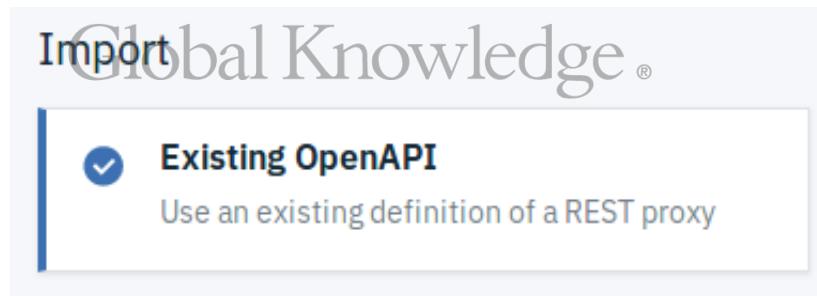


4.3. Import the API definition into the API Manager web application

In this section, you import the generated API definition file into the API Manager web application.

Although Loopback developers can work purely with the command-line interface, in this part you use a graphical editor to review the OpenAPI definition that is created for the Loopback application.

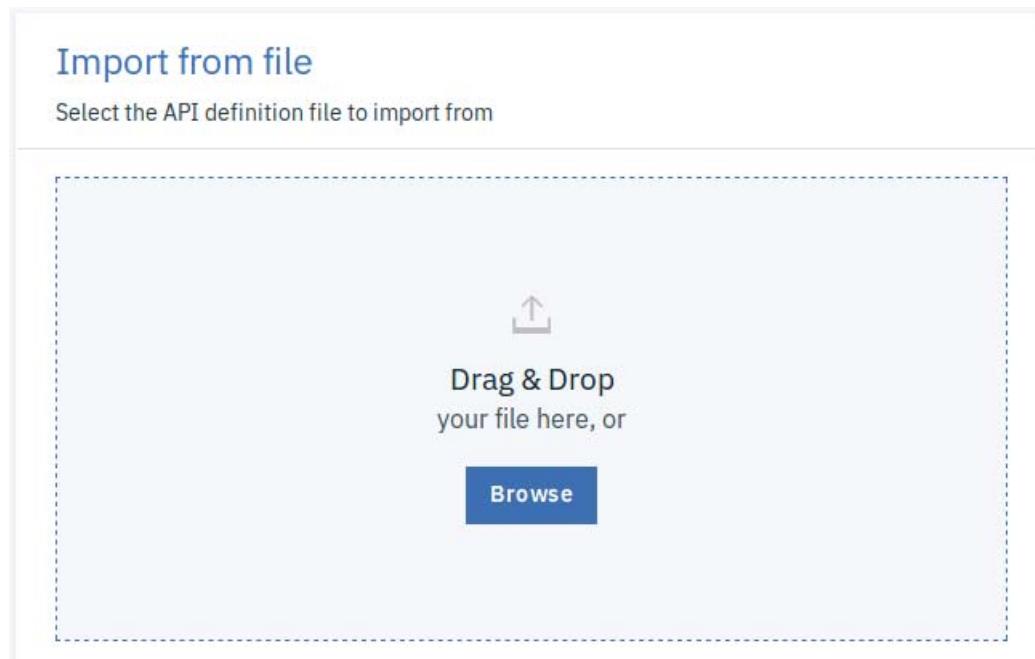
- 1. Start the API Manager web application.
 - a. Sign on to the API Manager web interface from a browser session. Type the address <https://manager.think.ibm>
 - b. Sign in with the user name and password that is associated with your API Manager account:
 - User name: ThinkOwner
 - Password: Passw0rd!
- Click **Sign in**.
- You are signed in to API Manager.
- 2. Import the OpenAPI API definition.
 - a. From the API Manager home page, click the **Develop APIs and Products** tile, or click **Develop** from the navigation bar.
 - b. From the Develop page, click the **Add** icon. Then, select **API**.
 - c. Click the tile to import an **Existing OpenAPI**.



The selector is displayed.

- d. Click **Next**.

- __ e. On the import from file page, click **Browse**.

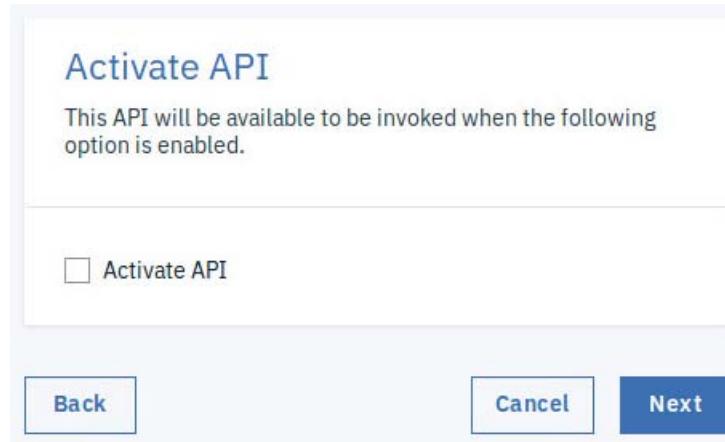


- __ f. Navigate to the ~/samples/notes directory, then, select the notes.yaml file. Click **Open**.
__ g. The YAML file is imported and successfully validated.

A screenshot of the "Import from file" page. The file "notes.yaml" is listed in the file selection area. A green circular icon with a checkmark is centered above the file list. At the bottom, a green notification bar says "YAML has been successfully validated" with a close button. Below the notification are "Cancel" and "Next" buttons.

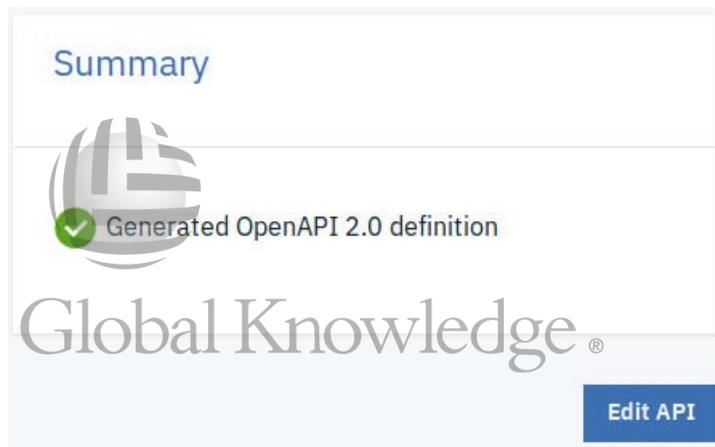
Click **Next**.

- __ h. Since you are going to test the application with the Loopback API Explorer, leave the Activate API cleared.



Click **Next**.

- __ i. After a few moments, the Summary page displays that the OpenAPI 2.0 definition is generated.



- __ j. Click **Edit API**.

The API opens in the Design view in API Manager.

4.4. Review the API definition in the API Manager web application

The **API definition** declares the interface of an API. The definition file specifies the API routes, methods, request, and response messages for each API operation. In this role, the API definition lists a set of operations that application developers can call.

The second role of the API definition file is to configure message processing rules and environment variables. The message processing policies specify how an API gateway transforms and routes API request and response messages. Environment variables set application server hosts.

In this section, you review the API definition that is generated by the Loopback scaffolding with the API Manager web application.

- 1. Review the 'notes' API definition on the API Editor Design tab.

The screenshot shows the API Manager interface with the 'notes' API definition. The top navigation bar includes 'Develop notes 1.0.0' and tabs for 'Design' (selected), 'Source', and 'Assemble'. The left sidebar lists sections: API Setup (Security Definitions, Security, Paths, Definitions, Properties, Target Services, Categories, Activity Log). The main content area is titled 'Info' with the sub-section 'Title' showing 'notes'. Below it are fields for 'Name' (set to 'notes') and 'Version' (set to '1.0.0'). A large watermark for 'Global Knowledge' is visible across the page.



In the **API Editor** view in API Manager, you can review and edit the API definition in 1 of 3 options:

- The **Design** tab opens the API definition document in a web form. The Design tab helps you enter information with the correct syntax.
- The **Source** tab opens the API definition file in the original text format. API definition file uses the YAML (yet another markup language) text format. The fields of the API definition file follow

the OpenAPI 2.0 specification. Switch to this view if you are familiar with the OpenAPI specification.

- The **Assemble** tab opens the message processing policy section of the API definition document in a graphical editor. You can also view the 'assembly' section of an API definition document on the **Source** tab.

- a. Examine the **API Setup** section of the 'Notes' API definition.

The Info section stores the name, version, and description of the API.

- b. Examine the **host** and **base path** section.

Host
The host used to invoke the API

Address (optional)

Base Path
The base path is the initial URL segment of the API to identify paths or operations

Base path (optional)
/api

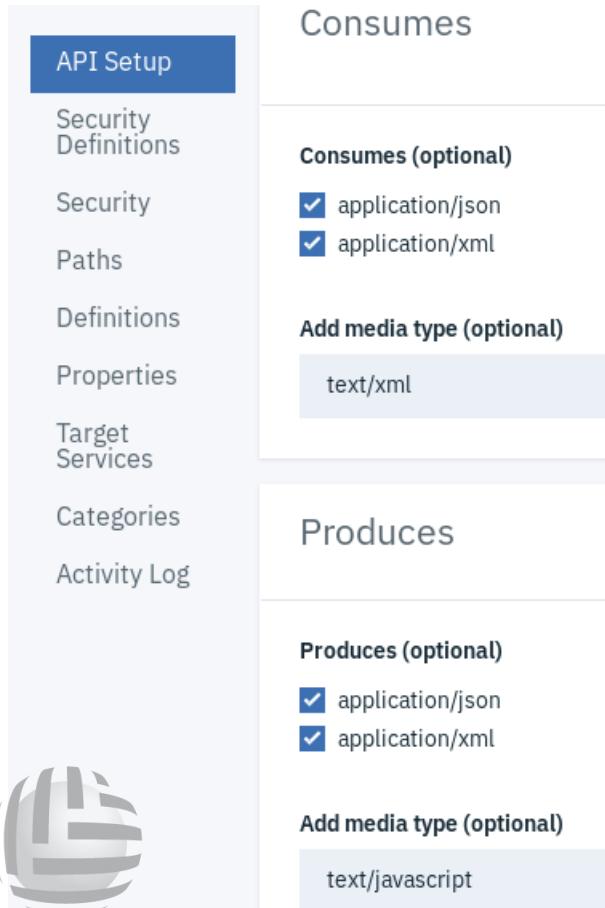


Information

The **host** property stores the host name or IP address of the server that hosts the published APIs. In an API Connect solution, the host variable is the address of the API Gateway. The **host** variable, which represents the entry point for the API, the API Gateway. The API Gateway address is used when the host variable is omitted.

The **base path** property sets the parent URL path for all API operations. In this example, the base path is set to '/api'.

- ___ c. Examine the **consumes** and **produces** section.



The screenshot shows the 'API Setup' interface. On the left, a sidebar lists navigation options: API Setup (selected), Security Definitions, Security Paths, Definitions, Properties, Target Services, Categories, and Activity Log. Below the sidebar is a large circular icon with a stylized 'GK' logo. The main area is divided into two sections: 'Consumes' and 'Produces'. The 'Consumes' section contains a heading 'Consumes (optional)' with two checked checkboxes: 'application/json' and 'application/xml'. Below this is a button labeled 'Add media type (optional)' with 'text/xml' listed. The 'Produces' section contains a heading 'Produces (optional)' with two checked checkboxes: 'application/json' and 'application/xml'. Below this is a button labeled 'Add media type (optional)' with 'text/javascript' listed.



Information

Global Knowledge®

The **consumes** and **produces** sections declare the media type in the API request message body and response message body. These two sections define how to encode an API request message, and what data formats to expect in the response message.

In this example, your application can send request messages with an 'application/json' or an 'application/xml' data format in the message body. If the API operation returns a response, the API implementation returns data in either format.

The consumes and produces section sets the default media type for all API operations. You can override the media type for a particular API operation in the paths section.

- ___ d. Click **Paths** in the selection area to examine the paths. Select the ellipsis in the /Notes path. Then, click **Edit**.

NAME	
/Notes	...
/Notes/replaceOrCreate	Edit
/Notes/upsertWithWhere	Delete
/Notes/{id}/exists	...
/Notes/{id}	...

- ___ e. The list of operations is displayed.

NAME	REQUIRED	LOCATED_IN	TYPE	DESCRIPTION	DELETE
/Notes	Edit		Delete	...	
POST	Edit	
PATCH	Edit	
PUT	Edit	
GET	Edit	



Information

The **paths** section lists the resource paths in the API. Each API operation consists of a resource path and an HTTP method. For example, the '**/notes**' path represents a notes data model object on the server.

- To create an instance of the notes object, send an HTTP **POST** request to **/notes**.
- To update the fields of a notes object, send an HTTP **PUT** or **PATCH** request to **/notes**.

- To retrieve the contents of a particular notes object, send an HTTP **GET** request to **/notes**.

-
- f. Click the return arrow or press **Cancel** to go back to the API definition.
 - g. In the **Definitions** section, select **Note**. Then, click **Edit** to review its properties.
 - h. In the edit definitions page, click **Navigate to Source View**.
The source page is displayed. Scroll down to view the definitions.



```

435 - text/javascript
436 definitions:
437   Note:
438     properties:
439       title:
440         type: string
441       content:
442         type: string
443       id:
444         type: number
445         format: double
446       required:
447         - title
448       additionalProperties: false
  
```



Information

The **definitions** section describes the structure of data objects that you send to or receive from API operation calls. For example, when you call GET /items, you retrieve a collection of 'Notes' objects from the server. The 'Notes' object consists of three properties: **title**, **content**, and **id**. The title and content properties are string data types, while the id property is a double data type. The id field is generated and not meant to be writeable. For more information, see ID properties on the page <https://loopback.io/doc/en/lb3/Model-definition-JSON-file.html>.

A one-to-one correlation exists between the structure of your LoopBack model objects and the data type definitions in this section of the API definitions file.



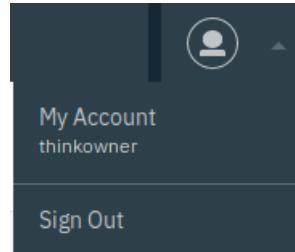
Information

The **Source** tab displays the API definition in its original format. The definition file follows a structured text format that is named YAML, or "yet another markup language". If you are familiar with the different sections of the OpenAPI specification, then the source format might be easier to view.

When you change values on the **Source** tab, the **Design** tab reflects those changes as well.

Keep in mind that the YAML file format and the OpenAPI specification are two separate topics. Not all YAML files are OpenAPI specifications. The YAML file format is an increasingly popular alternative to XML-based configuration files.

-
- ___ 2. Sign out of API Manager when you are finished reviewing the API definition.



4.5. Test the API operation with the LoopBack API Explorer

In this part, you test the API operations with the test feature of LoopBack, the API Explorer.

- 1. Start the notes Node application.

- a. From the /samples/notes directory, type:

```
npm start
```

- b. The output is displayed:

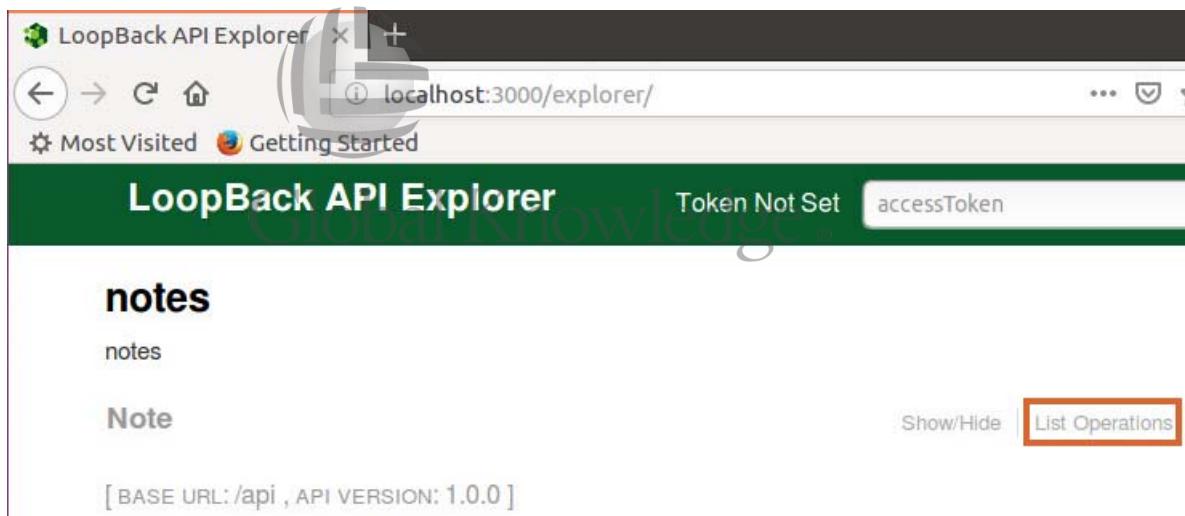
```
> notes@1.0.0 start /home/localuser/samples/notes
> node .

Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- c. Open the API Explorer in the browser. Type:

```
http://localhost:3000/explorer
```

The LoopBack API Explorer opens in the browser window.



- 2. Click **List operations**.

The list of operations is displayed.

- ___ 3. Test the **POST /notes** API operation from the Explorer page.
- ___ a. Click the **POST /Notes** operation in the operations list.

The screenshot shows the LoopBack API Explorer interface. At the top, it says "LoopBack API Explorer" and "Token Not Set". There is a text input field labeled "accessToken" and a green button labeled "Set Access Token". Below this, there is a section titled "notes" with a sub-section "notes". Under "notes", there is a table of operations:

Note		Show/Hide	List Operations	Expand Operations
PATCH	/Notes	Patch an existing model instance or insert a new one into the data source.		
GET	/Notes	Find all instances of the model matched by filter from the data source.		
PUT	/Notes	Replace an existing model instance or insert a new one into the data source.		
POST	/Notes	Create a new instance of the model and persist it into the data source.		

- ___ b. You see the example response of the call, and the parameter data for calling the POST operation with curl.



Global Knowledge ®

- ___ c. Click inside the example value area in the Parameters area on the page.

The screenshot shows a LoopBack API configuration interface for a POST request to the '/Notes' endpoint. The 'Example Value' for the 'data' parameter is highlighted with a red box. The example value is a JSON object:

```
{
  "title": "string",
  "content": "string",
  "id": 0
}
```

The 'Response Content Type' is set to 'application/json'. The 'Parameters' table shows the 'data' parameter with its description, type, and example value.

Parameter	Value	Description	Parameter Type	Data Type
data	<input type="text"/>	Model instance data	body	Model Example Value

The 'Example Value' column contains the JSON object shown above, with the 'title' and 'content' fields highlighted in yellow.

The example data is copied to the data area.

- ___ d. Change the values in the data area from "string" to some other values.

The screenshot shows the same LoopBack API configuration interface, but now with the 'data' parameter's example value changed. The 'Value' column for the 'data' parameter now contains a different JSON object:

```
{
  "title": "hello",
  "content": "world"
}
```

The 'Response Content Type' is still 'application/json'. The 'Parameters' table shows the 'data' parameter with its description, type, and updated example value.

Parameter	Value	Description	Parameter Type	Data Type
data	<input type="text"/>	Model instance data	body	Model Example Value

The 'Example Value' column now contains the JSON object shown above, with the 'title' and 'content' fields highlighted in yellow.

- ___ e. Click **Try it out**.

- ___ f. The curl command to call the operation is displayed and the response message is displayed.

```
Curl
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
  "title": "hello", \
  "content": "world" \
}' 'http://localhost:3000/api/Notes'

Request URL
http://localhost:3000/api/Notes

Response Body
{
  "title": "hello",
  "content": "world",
  "id": 1
}

Response Code
200
```

A note is added with a generated id of 1.

- ___ 4. Test the **GET /notes** API operation from the Explorer page.

- ___ a. Click the **GET /Notes** operation in the operations list.

Note

Show/Hide | List Operations | Expand Operations

PATCH /Notes Patch an existing model instance or insert a new one into the data source.

GET /Notes Find all instances of the model matched by filter from the data source.

PUT /Notes Replace an existing model instance or insert a new one into the data source.

POST /Notes Create a new instance of the model and persist it into the data source.

- ___ b. Click **Try it out**.

- ___ c. The note that you added earlier in the same test run is displayed.

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/Notes'
```

Request URL

```
http://localhost:3000/api/Notes
```

Response Body

```
[  
  {  
    "title": "hello",  
    "content": "world",  
    "id": 1  
  }  
]
```

Response Code

```
200
```

- ___ 5. Close the API Explorer in the browser.
___ 6. Stop the Node application by clicking CTRL-C in the terminal window.

End of exercise

Global Knowledge ®

Exercise review and wrap-up

In the first part of the exercise, you generated the scaffolding for the sample 'Notes' LoopBack application with the 'apic' command line utility. You reviewed the generated files, models, and data source. Next, you imported the OpenAPI definition for the application into API Manager and reviewed the definition with the graphical design editor.

In the second part of the exercise, you started and tested the LoopBack application with the LoopBack API Explorer page in the browser. You called API operations from the web-based API Explorer.



Exercise 5. Define LoopBack data sources

Estimated time

01:00

Overview

In this exercise, you bind the model to relational and non-relational databases with data sources. You define relationships between models. Finally, you test API operations with the LoopBack API Explorer.

Objectives

After completing this exercise, you should be able to:

- Install and configure the MySQL connector
- Install and configure the MongoDB connector
- Generate models and properties from a data source
- Define relationships between models
- Test an API with the LoopBack API Explorer

Introduction

The Node.js runtime environment consists of an interpreter for the JavaScript programming language, and a library to build server-side web applications. By its design, the library is minimalist. You install the software packages that you choose to build your application.

The LoopBack framework provides a robust set of libraries to quickly build REST APIs based on model objects. In the previous exercise, you examined how to generate, customize, and test a LoopBack application that saved its data in memory.

In this exercise, you build a second application that persists its model data to data stores: MySQL and MongoDB. The MySQL database is a relational database that holds its data in tables. The MongoDB database is a document-based non-relational database: it stores its data in documents that do not conform to a set schema.

Requirements

You must complete this lab exercise in the student development workstation: the Ubuntu host environment. This virtual machine is pre-configured with the Node runtime environment, the 'npm' Node package manager, and the IBM API Connect toolkit. Dependencies exist in the course

exercise from this lab onward. You must complete this exercise to get the inventory case study to work.



Exercise instructions

Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



Global Knowledge®

5.1. Create the inventory application and MySQL data source

In this section, create the directory structure and configuration files for the Inventory LoopBack application. This application manages a set of descriptions for memorable items from IBM's history. The LoopBack framework provides REST API access to the inventory items through the model class.

- ___ 1. Open the terminal emulator application.
 - ___ a. From the start menu, click the Terminal from the applications menu.
- ___ 2. Create a directory for the application, named `inventory`, in the student home directory.
 - ___ a. In the home directory, create a directory that is named `inventory`.

```
$ mkdir ~/inventory
$ cd inventory
$ pwd
/home/localuser/inventory
```

- ___ 3. Generate the directory structure and configuration files for an empty server LoopBack application.
 - ___ a. Run the `apic lb` command in the `inventory` directory.
 - ___ b. Create an empty-server LoopBack application.
- ___ c. Confirm that the 'apic' utility generated the LoopBack application successfully.

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

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

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

- ___ 4. Install the LoopBack MySQL connector into the `inventory` application.

- ___ a. Install the `loopback-connector-mysql` npm package.

```
$ npm install --save loopback-connector-mysql
```

- ___ b. Examine the dependencies section of the project manifest file.

```
$ more package.json
...
"dependencies": {
  "compression": "^1.0.3",
  "cors": "^2.5.2",
  "helmet": "^3.10.0",
  "loopback": "^3.22.0",
  "loopback-boot": "^2.6.5",
  "loopback-component-explorer": "^6.2.0",
  "loopback-connector-mysql": "^5.3.1",
  "serve-favicon": "^2.0.1",
  "strong-error-handler": "^3.0.0"
},
...
...
```



Information

The LoopBack connector is a software package that creates a data source: a Node.js object that manages access from the LoopBack framework to a data store.

Before you begin, you must download and install the correct connector for your data source. Use the 'npm' utility to retrieve the most recent version of the LoopBack connector.

- ___ 5. Define a LoopBack data source that is named `mysql-connection` with the 'apic' command line utility.

- ___ a. Create a data source object with the apic command line utility.

```
$ apic lb datasource
```

- ___ b. Enter the following properties for a data source that is named `mysql-connection`:

- Data source name: `mysql-connection`
- Connector: MySQL (supported by StrongLoop)
- Connection String url to override other settings: Leave blank (Enter)
- Connector-specific configuration:
 - o Host: `platform.think.ibm`
 - o Port: `3306`
 - o User: `localuser`
 - o Password: `passw0rd`
 - o Database: `think`

```

? Enter the data-source name: mysql-connection
? Select the connector for mysql-connection:
> MySQL (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings: <leave blank>
? host: platform.think.ibm
? port: 3306
? user: localuser
? password: passw0rd
? database: think

```

- ___ c. Review the changes to the server/datasources.json configuration file.

```

$ more server/datasources.json
{
  "mysql-connection": {
    "host": "platform.think.ibm",
    "port": 3306,
    "url": "",
    "database": "think",
    "password": "passw0rd",
    "name": "mysql-connection",
    "user": "localuser",
    "connector": "mysql"
  }
}

```



Information

Global Knowledge®

The apic lb datasource command adds a data source definition to your LoopBack application. The data source relies on the LoopBack connector: a npm package that implements the data source object. You can modify the settings directly in the server/datasources.json configuration file.

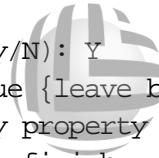
5.2. Create the item LoopBack model

The `item` model object represents a JSON representation of the item table from the **think** MySQL database. In this section, define the properties in the item model. Create the model. Then, define the model properties to map the LoopBack model to the item table in the MySQL database.

- __ 1. Create a LoopBack model named `item`.
 - __ a. Ensure that you are in the `/home/localuser/inventory` directory.
 - __ b. From the terminal, type:
`apic lb model`
 ? Enter the model name: `item`
 > Select the datasource to attach item to: `mysql-connection (mysql)`
 ? Select model's base class
 > `PersistedModel`
 ? Expose item via the REST API: `Y`
 ? Custom plural form: (Press Enter)
 ? Common model or server only?
 > `common`
 ? Property name: `name`
 ? Property type:
 > `string`
 ? Required? (y/N): `Y`
 ? Default value {leave blank for none}:
 Enter an empty property name when done.
 Press enter to finish.



Information



Global Knowledge®

You created a model that is named `item` with a single property that is named `name`. You might add further properties when you create the model. Instead, you review the files that are generated.

Then, you copy the remaining model properties into the `item.json` file that is created in the `/inventory/common/models` directory. You can also add properties to a model with the `apic lb` property command. The final list of properties is shown in the table.

Table 5. *Item* model properties

Required	Property name	Type	Description
Yes	<code>name</code>	<code>string</code>	
Yes	<code>description</code>	<code>string</code>	item description
Yes	<code>img</code>	<code>string</code>	location of item image
Yes	<code>img_alt</code>	<code>string</code>	item image title
Yes	<code>price</code>	<code>number</code>	item price
No	<code>rating</code>	<code>number</code>	item rating

__ 2. Review the generated files for the `item` model.

__ a. Change directory to `~/inventory/common/models`.

```
$ cd ~/inventory/common/models
$ pwd
/home/localuser/inventory/common/models
```

__ b. The files that are named `item.js` and `item.json` were created during model generation. Review the `item.js` file.

```
$ more item.js
'use strict';
module.exports = function(Item) {
};
```

__ c. Review the `item.json` file.

```
$ more item.json
{
  "name": "item",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "name": {
      "type": "string",
      "required": true
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```

__ a. Change directory to `~/inventory/server`.

```
$ cd ~/inventory/server
$ pwd
/home/localuser/inventory/server
```

- ___ b. The file that is named `model-config.json` was created when the application was created and updated during model generation.

```
$ more model-config.json
{
  "_meta": {
    "sources": [
      "loopback/common/models",
      "loopback/server/models",
      "../common/models",
      "./models"
    ],
    "mixins": [
      "loopback/common/mixins",
      "loopback/server/mixins",
      "../common/mixins",
      "./mixins"
    ]
  },
  "item": {
    "dataSource": "mysql-connection",
    "public": true
  }
}
```

- ___ 3. Update the `item.json` file with the remaining properties.

- ___ a. Change directory to `~/inventory/common/models`.

```
$ cd ~/inventory/common/models
$ pwd
/home/localuser/inventory/common/models
```

- ___ b. Move the file that is named `allitem.json` to the `models` directory.

```
$ mv ~/lab_files/datasources/allitem.json item.json
```

- __ c. Review the changed item.json file.

```
$ cat item.json

{
  "name": "item",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "name": {
      "type": "string",
      "required": true
    },
    "description": {
      "type": "string",
      "required": true,
      "description": "item description"
    },
    "img": {
      "type": "string",
      "required": true,
      "description": "location of item image"
    },
    "img_alt": {
      "type": "string",
      "required": true,
      "description": "item image title"
    },
    "price": {
      "type": "number",
      "required": true,
      "description": "item price"
    },
    "rating": {
      "type": "number",
      "required": false,
      "description": "item rating"
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```

-
- 4. The item model now has all the properties defined.



Information

The LoopBack framework has a feature to discover models from relational tables. This feature is available in earlier versions of the API Designer that comes with the API Connect toolkit. At course development time, the API Designer did not fully support building Loopback applications. Students use the command line interface to build LoopBack applications in this class. Advanced students might try building a script that uses the LoopBack APIs to discover model data from relational tables.

Refer to the LoopBack documentation at

<https://loopback.io/doc/ja/lb3/Implementing-model-discovery.html>



Global Knowledge®

5.3. Test the inventory application and items model

In this section, start a local copy of the Inventory LoopBack application in the terminal. Examine the API operations for the item model in the LoopBack API Explorer view. Test the API operations and retrieve model data through the mysql-connection data source.

- 1. Open a terminal window, or use the currently open terminal.

- a. Navigate to the inventory application directory.

```
$ cd ~/inventory  
$ pwd  
/home/localuser/inventory
```

- b. Start the Loopback application.

```
$ npm start  
> inventory@1.0.0 start /home/localuser/inventory  
> node .
```

Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer

- c. Open the API Explorer in the browser. Type:
<http://localhost:3000/explorer>

The LoopBack API Explorer opens in the browser window.

- 2. Click **Show/Hide**.

The list of operations is displayed.

The screenshot shows the LoopBack API Explorer interface. At the top, there's a browser header with the URL `localhost:3000/explorer/#/item`. Below it, a green navigation bar says "LoopBack API Explorer". On the right of the bar are buttons for "Token Not Set" and "accessToken", and a "Set Ac" button. The main content area has a title "inventory" and a sub-section "inventory". Under "inventory", there's a section titled "item" with the following operations listed:

- PATCH /items**: Patch an existing model instance or insert a new one into the data source.
- GET /items**: Find all instances of the model matched by filter from the data source.
- PUT /items**: Replace an existing model instance or insert a new one into the data source.
- POST /items**: Create a new instance of the model and persist it into the data source.
- PATCH /items/{id}**: Patch attributes for a model instance and persist it into the data source.
- GET /items/{id}**: Find a model instance by {{id}} from the data source.

___ 3. Retrieve a list of inventory items with the GET /items operation.

___ a. Click the GET /items operation in the operations list.

This screenshot shows the same LoopBack API Explorer interface as above, but with a focus on the "item" operations. The "GET /items" operation is highlighted with a red border. The other operations are shown in a standard list format.

___ b. You see the sample response of the call, and the parameter filter for calling the GET operation with curl.

___ c. Click **Try it out**.

- ___ d. You see the generated curl command, the response body, and the response code.

```
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/items'
```

Request URL

```
http://localhost:3000/api/items
```

Response Body

```
[
  {
    "name": "Dayton Meat Chopper",
    "description": "Punched-card tabulating machines and time clocks were not the only products",
    "img": "images/items/meat-chopper.jpg",
    "img_alt": "Dayton Meat Chopper",
    "price": 4599.99,
    "rating": 31.32,
    "id": 1
  },
  {
    "name": "Hollerith Tabulator",
    "description": "This equipment is representative of the tabulating system invented and built by Herman Hollerith. It was used to process the 1890 U.S. Census data. The machine used punched cards to store data and could sort, tabulate, and print results. It was the first large-scale automatic data processing system and paved the way for modern computing.",
    "img": "images/items/hollerith-tabulator.jpg",
    "img_alt": "Hollerith Tabulator",
    "price": 10599.99,
    "rating": 0,
    "id": 2
  }
]
```

Response Code



```
200
```

The node application successfully returns the inventory items by calling the MySQL database with the mysql-connector.

- ___ e. Review the request and response headers from the API call.

Request

```
curl -X GET --header 'Accept: application/json'
'http://localhost:3000/api/items'
```

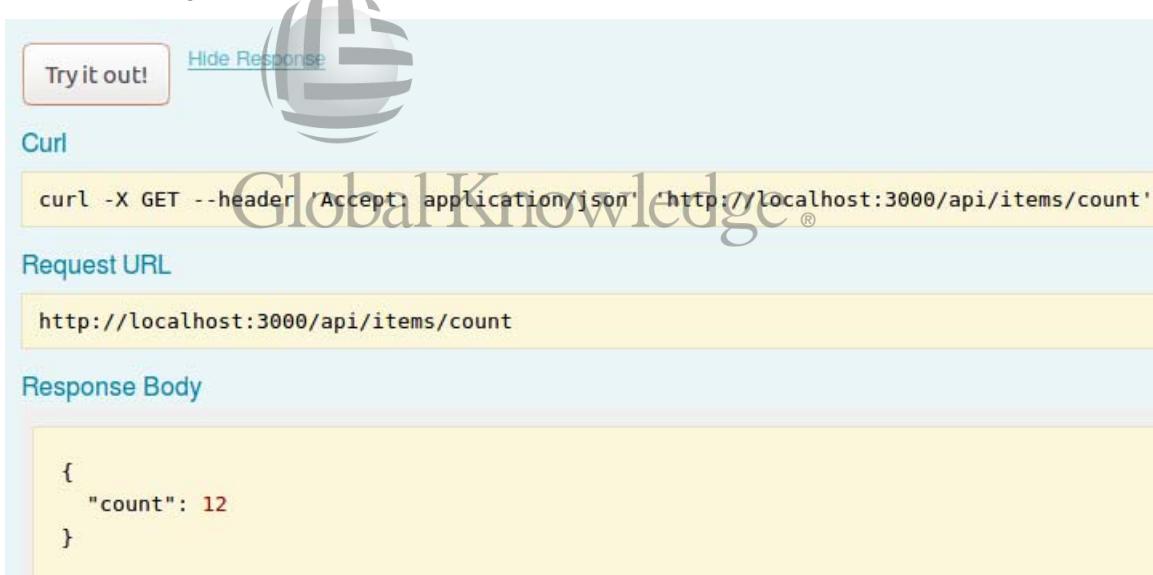
Response

Code: 200

- ___ f. Review the list of items in the message body.

```
[
  {
    "description": "Punch-card tabulating machines and...",
    "img": "images/items/meat-chopper.jpg",
    "img_alt": "Dayton Meat Chopper",
    "name": "Dayton Meat Chopper",
    "price": 4599.99,
    "rating": 31.32,
    "id": 1
  },
  ...
]
```

- ___ 4. Check the number of items in the think MySQL database with the GET /items/count API operation.
- ___ a. In the LoopBack API Explorer page, click **List Operations** operation to display the list of operations.
- ___ b. Scroll down in the page. Then, select GET /items/count in the list of operations.
- ___ c. Click **Try it out**.



The operation returns the count of the number of rows in the think MySQL database.

- ___ d. Close the Loopback API Explorer in the browser.
- ___ e. Click **CTRL-C** to stop the inventory application that is running in the terminal.

5.4. Create a MongoDB data source and the review model

In this section, you define a second model object, named `ratings`, that represent user reviews on your inventory API. Since people write reviews of varying lengths, it makes sense to use a document-based database instead of a relational database to save the reviews.

The MongoDB database is an example of a document-centric, non-relational database. Instead of storing data in a database table with defined columns, MongoDB stores each record as a JSON file of data.

At the end of this section, you connect the `ratings` LoopBack model object to a MongoDB data source. The data source sends and retrieves user reviews from a MongoDB database. In the next exercise, you add logic to the Node application to calculate the average rating, each time a user types a rating change for inventory items.



Information

One of the powerful features of LoopBack is that it can model relationships between relational tables and unstructured, non-relational data. After you define the mongoDB model, you create a foreign key relationship between the item MySQL table and the MongoDB review data structure.

For more information on the content of these two databases, see the information section at the end of this exercise.

- ___ 1. Open a terminal emulator window, or use a currently open terminal.
- ___ 2. Define a MongoDB data source in the inventory application.

- ___ a. Verify that the current directory is the inventory application directory.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- ___ b. Run the apic data source command.

```
$ apic lb datasource
```

- ___ c. Create a data source that is named `mongodb-connection` with the MongoDB LoopBack connector.

- Data source name: `mongodb-connection`
- Connector: MongoDB (supported by StrongLoop)
- Connector-specific configuration:
 - o Host: `platform.think.ibm`
 - o Port: `27017`
 - o User: `<leave blank>`
 - o Password: `<leave blank>`

- Database: think
- Install loopback-connector-mongodb: Y

? Enter the data-source name: mongodb-connection
? Select the connector for mongodb-connection:
> MongoDB (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database): Enter
? host: platform.think.ibm
? port: 27017
? user:
? password:
? database: think
? Install loopback-connector-mongodb@^3.6.0 (Y/n) Y



Information

When the apic lb datasource command defines a data source, it checks whether you have the correct LoopBack connector for the data source. If you did not install the connector, you can select yes to the install loopback-connector-mongodb prompt. This command is the same as running the npm install command:

```
npm install --save loopback-connector-mongodb
```

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

— 3. Create a LoopBack model object that is named `review` to represent reviews that are stored in the MongoDB database.

— a. Run the apic command to create a LoopBack model.

```
$ apic lb model
```

— b. Define a model that is named `review`, with the following properties:

- Model name: `review`
- Attach model to data source: `mongodb-connection` (`mongodb`)
- Base model class: `PersistedModel`
- Expose `review` via the REST API? N
- Common model or server only? common

? Enter the model name: **review**
 ? Select the data-source to attach review to:
 > **mongodb-connection (mongodb)**
 ? Select models base class:
 > **PersistedModel**
 ? Expose review via the REST API? (Y/n): **N**
 ? Custom plural form: (Press Enter)
 ? Common model or server only?
 > **common**

**Note**

Remember to answer **No** in the expose review via the REST API question. You do not expose the review model in the inventory API. You write custom code in a later exercise that modifies review records in a remote method.

- ___ c. Add a property that is named **date**, of type **date**, to the apic model.

Enter an empty property name when done.

? Property name: **date**
 ? Property type:
 > **date**
 ? Required? **Y**
 ? Default value [leave blank for none]: <leave blank>

- ___ d. Enter the remaining properties according to the table.

Global Knowledge®
Table 6. Review model properties

Property name	Property type	Required	Default value
date	date	Yes	<leave blank>
reviewer_name	string	No	<leave blank>
reviewer_email	string	No	<leave blank>
comment	string	No	<leave blank>
rating	number	Yes	<leave blank>

- ___ e. Leave the property name blank after you created the last property.

Let's add another review property.

Enter an empty property name when done.

? Property name:
 localuser@ubuntu:~/inventory\$

5.5. Define a relationship between the item and review models

In the LoopBack framework, you can define relationships between two sets of models in your application. When you create a relationship, you impose a set of behaviors on a pair of model objects. For example, an online product review does not exist on its own. Every review applies to exactly one item in the inventory database.

To model the relationship in the inventory LoopBack application, you create a relationship that states an `item` has many `reviews`. Use the ‘apic’ command line utility to define the relationship, and save the information in the model configuration files. Ensure that you are in the `~/inventory` directory.

- ___ 1. Define a one-to-many relationship between an `item` and `review` models.

- ___ a. Start the apic LoopBack relationship command.

```
$ apic lb relation
```

- ___ b. Create a model relationship named `reviews`: a one-to-many relationship from the `item` model to the `review` model.
 - Create a relationship from: `item`
 - Relation type: `has many`
 - Create a relationship with: `review`
 - Property name for the relation: `reviews`
 - Custom foreign key: `<leave blank>`
 - Require a through model: `No`

? Select the model to create the relationship from:

> `item`

? Relation type:

> `has many`

? Choose a model to create a relationship with:

> `review`

? Enter the property name for the relation: `reviews`

? Optionally enter a custom foreign key: `<leave blank>`

? Require a through model? `No`

? Allow the relation to be nested in REST APIs? `No`

? Disable the relation from being included? `No`

- __ c. Review the generated `review.json` model definition file.

```
$ cat common/models/review.json
```

```
{  
  "name": "review",  
  "base": "PersistedModel",  
  "idInjection": true,  
  "options": {  
    "validateUpsert": true  
  },  
  "properties": {  
    "date": {  
      "type": "date",  
      "required": true  
    },  
    "reviewer_name": {  
      "type": "string"  
    },  
    "reviewer_email": {  
      "type": "string"  
    },  
    "comment": {  
      "type": "string"  
    },  
    "rating": {  
      "type": "number",  
      "required": true  
    }  
  },  
  "validations": [],  
  "relations": {},  
  "acls": [],  
  "methods": {}  
}
```

- __ d. Open the `item.json` model definition file.

```
$ more common/models/item.json
```

- ___ e. Review the relationship definition section in `item.json`.

```
"validations": [ ],
  "relations": {
    "reviews": {
      "type": "hasMany",
      "model": "review",
      "foreignKey": ""
    }
  },
  "acls": [],
  "methods": {}
```



Note

The relationships that you create in your LoopBack application are logical relationships. The underlying MySQL and MongoDB databases *do not* have a relationship between its records.

From the point of view of the `item` table in the MySQL database, `reviews` is a property that stores an identifier value. This identifier is a number that corresponds to a document in the `review` MongoDB database.

The API operations in your inventory LoopBack application enforce the `reviews` relationship. It controls how users can create, update, and delete `item` and `review` model objects.



Global Knowledge®

5.6. Test the inventory application and review model

In this section, start a local copy of the Inventory LoopBack application in the terminal. Examine the API operations for the item model in the LoopBack API Explorer view. Test the API operations and retrieve model data through the mysql-connection and the mongodb-data source.

- 1. Open a terminal window.

- a. Navigate to the inventory application directory.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- b. Start the Loopback application.

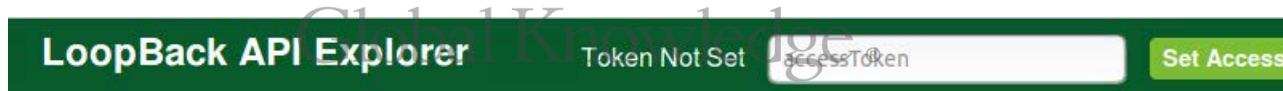
```
$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .
```

Web server listening at: http://localhost:3000
 Browse your REST API at http://localhost:3000/explorer

- c. Open the API Explorer in the browser. Type:

<http://localhost:3000/explorer>

The LoopBack API Explorer opens in the browser window.



inventory

inventory

item

Show/Hide | List Operations | Expand Operations

[BASE URL: /api , API VERSION: 1.0.0]

Notice that you selected 'N' when prompted earlier whether you wanted the review model to be exposed as REST API, so you don't see any review operations in the API Explorer.

- 2. Click **Show/Hide**.
- 3. Earlier you tested the GET /items operation that returned data from the MySQL table with the id field that has a range 1 - 12.
- 4. Although you did not see any REST operations for the review model, you can run queries from the list of item operations that returns the review data based on the relationship with the item table.

- ___ 5. Scroll down in the API Explorer until the reviews are displayed.

GET	/items/{id}/reviews	Queries reviews of item.
POST	/items/{id}/reviews	Creates a new instance in reviews of this model.
DELETE	/items/{id}/reviews	Deletes all reviews of this model.
GET	/items/{id}/reviews/{fk}	Find a related item by id for reviews.
PUT	/items/{id}/reviews/{fk}	Update a related item by id for reviews.
DELETE	/items/{id}/reviews/{fk}	Delete a related item by id for reviews.
GET	/items/{id}/reviews/count	Counts reviews of item.

- ___ 6. Retrieve a list of reviews for the items with the `id` value of 1.

- ___ a. Click the GET `/items/{id}/reviews` in the operations list.

GET	/items/{id}/reviews	Queries reviews of item.
-----	---------------------	--------------------------

- ___ b. You see the sample response of the call, and the parameter filter for calling the GET operation with curl.
 ___ c. Notice that the `id` is a required parameter. Type the value 1 in the `id` parameter area.

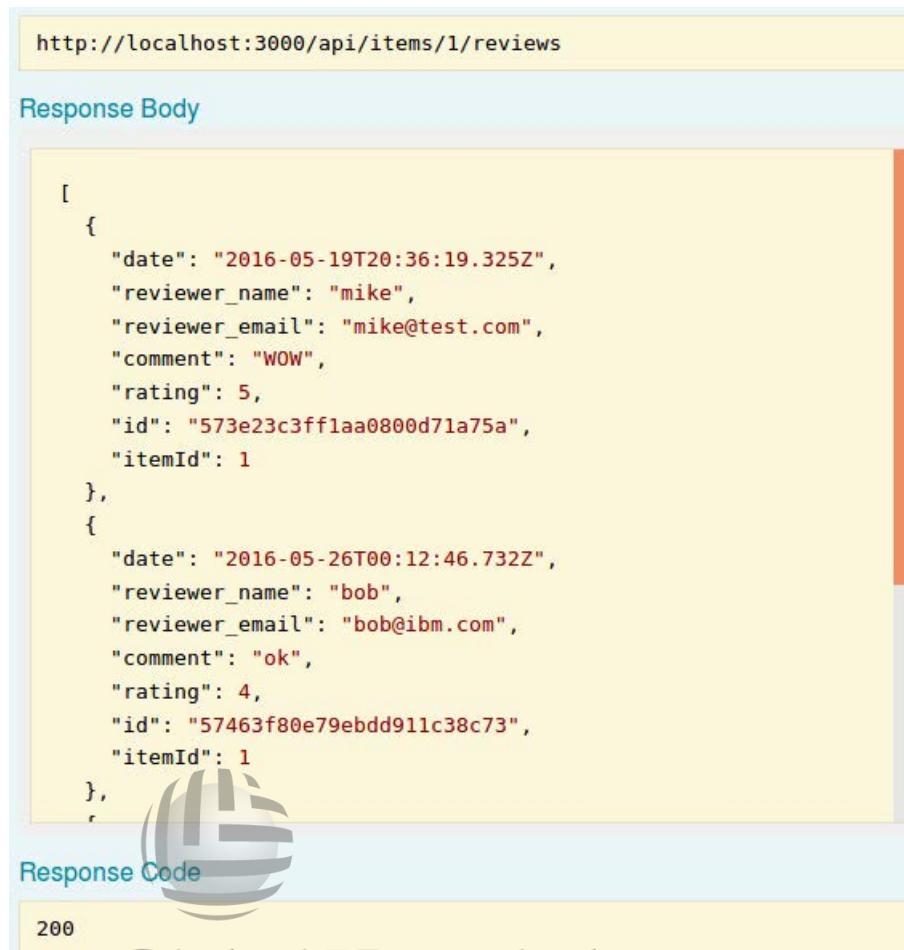
Parameters

Parameter	Value	Description	Parameter Type	Data Type
<code>id</code>	(required)	item id	path	string
<code>filter</code>	<input type="text"/>		query	string

Try it out!

Click **Try it out**.

- d. The response is displayed in the API Explorer.



```

http://localhost:3000/api/items/1/reviews

Response Body

[{"date": "2016-05-19T20:36:19.325Z", "reviewer_name": "mike", "reviewer_email": "mike@test.com", "comment": "WOW", "rating": 5, "id": "573e23c3ff1aa0800d71a75a", "itemId": 1}, {"date": "2016-05-26T00:12:46.732Z", "reviewer_name": "bob", "reviewer_email": "bob@ibm.com", "comment": "ok", "rating": 4, "id": "57463f80e79ebdd911c38c73", "itemId": 1}]

Response Code
200

```

Two ratings are associated with the item id with a value 1. The itemId is a foreign key value in the reviews collection.

- e. The results confirm that your inventory LoopBack node application is working. The application uses two data sources and extracts data from two separate databases based on the relationship that you set up between the items and reviews.
- f. Close the Loopback API Explorer in the browser.
- g. Click CTRL-C to stop the inventory application that is running in the terminal.

End of exercise

Exercise review and wrap-up

The first part of the exercise examined how to generate a LoopBack application with the API Connect Toolkit. The ‘apic lb’ command creates a starting point for an empty LoopBack API application in the workstation. You defined two business models in LoopBack: an ‘item’ record and a ‘review’ document.

In the second part of the exercise, you configured LoopBack data types for two databases. A MySQL relational database and a document-based MongoDB database. You defined a relationship that linked ‘item’ MySQL table with ‘review’ documents in the MongoDB database.

In the last part, you ran queries against both databases based on the relationship you defined in LoopBack.

This exercise uses a MySQL relational database and a mongo database with unstructured data. The MySQL service and the mongod service should start automatically when the image is started.



Information

MySQL:

Verify that MySQL is started with the command:

```
$ sudo /etc/init.d/mysql status
password for localuser: passw0rd
mysql.service - MySQL Community Server
Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
Active: active (running)

MySQL sign on and queries.

$ mysql -u localuser -p
Enter password: passw0rd
mysql> SHOW DATABASES;
mysql> USE think;
mysql> SHOW TABLES;
Tables in think
item
mysql> SELECT COUNT(*) FROM item;
COUNT
12
mysql> exit;
```

mongodb:

Verify that mongodb is started with the command:

```
$ sudo /etc/init.d/mysql status  
password for localuser: passw0rd  
mongod.service - MongoDB Database Server  
Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset;  
enabled)  
Active: active (running)
```

mongo sign on and queries.

```
$ mongo  
> show dbs  
admin  
config  
local  
think  
> use think  
> show collections  
review  
> coll = db.review  
think.review  
> coll.find();  
displays json table  
> coll.count();  
6  
> exit  
bye
```



Global Knowledge®

Exercise 6. Implement event-driven functions with remote and operation hooks

Estimated time

00:45

Overview

In this exercise, you add custom logic to the generated LoopBack models. You extend the inventory model with remote hooks: preprocessing and post-processing functions that change the behavior of API operation calls. You also create two operation hooks: event listeners that the LoopBack framework triggers when it accesses or modifies persisted model objects.

Objectives

After completing this exercise, you should be able to:

- Define an operation hook in a LoopBack application
- Define a remote hook in a LoopBack application
- Test remote and operation hooks in the LoopBack Explorer

Introduction

In the previous exercise, you created the inventory LoopBack application. You generated a set of API operations based on two model objects: item and review. The item model represents details of items in the think MySQL database. The review model stores user-submitted reviews of items for sale. A non-relational, document-centric MongoDB database stores the reviews.

LoopBack has three ways to add application logic to models: adding remote methods, remote hooks, and operation hooks.

In this exercise, you extend the inventory model with remote hooks: preprocessing and post-processing functions that change the behavior of API operation calls.

You also create two operation hooks: event listeners that the LoopBack framework triggers when it accesses or modifies persisted model objects.

Requirements

Before you start this exercise, you must complete the data sources exercise in this course.

You must complete this lab exercise on the remote student image: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



Exercise instructions

Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



6.1. Modify the API base path

By default, LoopBack applications host API operations with the `/api` base path. Change the base path to: `/inventory`

- ___ 1. Open the inventory application in the terminal.
 - ___ h. Verify that the current directory is the inventory application directory.
- ```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```
- \_\_\_ 2. Edit the API root in the `server/config.json` configuration path to: `"/inventory"`
  - \_\_\_ a. From the terminal, type `gedit server/config.json`.  
The file opens in the editor.
  - \_\_\_ b. In the `"restApiRoot"` property, change the value from `"/api"` to `"/inventory"`



- \_\_\_ c. Save your changes.
- \_\_\_ 3. The base path value contains this value when you generate the OpenAPI definition for the inventory application. You also see this value in request path when you test the application.

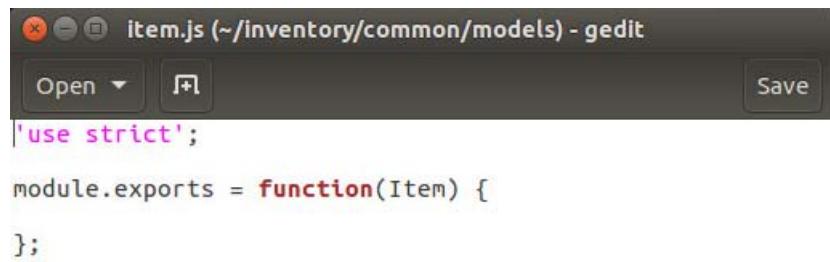
## 6.2. Create an operation hook

An operation hook is a function that listens to an entire class of operations: create, read, update, or delete operation. Whereas remote hooks run before or after a specific named operation, operation hooks listen to a class of operations for a model that is persisted in a data source.

- The `access` hook is triggered whenever the LoopBack framework queries a data source for models, for example, when you call any API operation that runs the `create`, `retrieve`, `update`, or `delete` method in the `PersistedModel` object.
- The `before save` hook is triggered before any operation that creates or updates a `PersistedModel` object.
- The `after save` hook is triggered after any operation that creates or updates a `PersistedModel` object.
- The `before delete` and `after delete` hooks are triggered before LoopBack removes a model from a data source. Specifically, these hooks run when you call the `deleteAll`, `deleteById`, and `prototype.delete()` functions in the `PersistedModel` object.
- The `loaded` hook is triggered after a LoopBack connector retrieves model data, but before it creates a model object from the data. You can use a loaded hook to transform or decrypt raw data from a connector before LoopBack builds a model instance.
- The `persist` hook listens to operations that save data to the data source. While the `before save` hook listens to changes to a LoopBack model object, the `persist` hook runs before the LoopBack framework saves, or persists, a modified model object to the data source.

In this section, you define an `access` hook to listen to items that are retrieved from a data source. You also monitor updates to the item model object with a `before save` hook. For example, you can log whether the LoopBack framework created or updated a persisted model object.

1. Open the `common/models/item.js` model script file from the terminal.
  - a. `$ cd common/models` directory.
  - b. Open the `item.js` script in an editor.



```
item.js (~/inventory/common/models) - gedit
Open Save
'use strict';

module.exports = function(Item) {
};
```

2. Create an access hook in the `item` model.
  - a. Open the `item.js` model script file in the editor.

- \_\_ b. Define a listener function that observes the access event.

```
Item.observe('access', function logQuery(ctx, next) {
 console.log('Accessed %s matching %j',
 ctx.Model.modelName, ctx.query.where);
 next();
});
```



### Information

The LoopBack framework calls the `logQuery` function when an API operation queries information from a data source. The function writes the name of the model and the lookup parameters in the console log.



Global Knowledge®

3. Create an `after save` hook in the item model.

a. In the `item.js` script file, add a second listener function that observes the `after save` event.

```
Item.observe('after save', function(ctx, next) {
 if (ctx.instance) {
 console.log(
 'Saved %s#%s',
 ctx.Model.modelName, ctx.instance.id);
 } else {
 console.log(
 'Updated %s matching %j',
 ctx.Model.plural modelName, ctx.where);
 }
 next();
});
```

```
'use strict';

module.exports = function(Item) {
 Item.observe('access', function logQuery(ctx, next){
 console.log('Accessed %s matching %j',
 ctx.Model.modelName, ctx.query.where);
 next();
 });

 Item.observe('after save', function(ctx, next) {
 if (ctx.instance) {
 console.log(
 'Saved %s#%s',
 ctx.Model.modelName, ctx.instance.id);
 } else {
 console.log(
 'Updated %s matching %j',
 ctx.Model.plural modelName, ctx.where);
 }
 next();
 });
};
```



### Information

The LoopBack framework runs the `after save` operation hook when an API operation creates or updates a record in the data source. The `ctx.instance` object represents the newly created model object. If the data source did not create a model instance, the log records the existing object that the framework updated.

4. Save the `item.js` model script file.

## 6.3. Test the operation hook

To test the access operation hooks, run the inventory API application from the terminal. Test API operations in the API Explorer and review the entries in the application log.

- 1. Open a terminal window.

- a. Confirm that the current directory is in the inventory application.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- b. Start the Loopback application.

```
$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .

Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- c. Open the API Explorer in the browser. Type:

<http://localhost:3000/explorer>

- d. The LoopBack API Explorer opens in the browser window.

- 2. Retrieve the count of the items in the API Explorer.

- a. Click the GET /items/count in the operations list.

- b. You see the sample response of the call, and the parameter filter for calling the GET operation with curl.

- c. Click **Try it out**.

- d. The result is displayed in the API Explorer.

[Try it out!](#)

[Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/inventory/items/count'
```

Request URL

```
http://localhost:3000/inventory/items/count
```

Response Body

```
{
 "count": 12
}
```

- \_\_\_ 3. Review the output from the operation hook in the application log.
  - \_\_\_ a. Confirm that the **access** operation hook logged the query against the **item** database.
  - \_\_\_ b. Return to the terminal window where the application was started.

```
localuser@ubuntu:~/inventory$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .

(node:56438) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
Accessed item matching {}
```

The line is displayed:

Accessed item matching {}

- \_\_\_ 4. Go to the LoopBack Explorer browser window.
  - \_\_\_ 5. Test the GET /items/{id} API operation with an "id": 1.
    - \_\_\_ a. In the API Explorer, click the GET /items/{id} operation.
    - \_\_\_ b. In the id parameter, type the value 1 as the query parameter.
    - \_\_\_ c. Click **Try it out**.
    - \_\_\_ d. Confirm that a value response object is returned.
    - \_\_\_ e. Also, confirm that the access operation hook logged the data source retrieval and query parameter in the terminal console.
- Browse your REST API at http://localhost:3000/explorer  
 Accessed item matching {}  
 Accessed item matching {"id":1}
- \_\_\_ f. Close the Loopback API Explorer in the browser.
  - \_\_\_ g. Click CTRL-C to stop the inventory application that is running in the terminal.

## 6.4. Create a remote hook

A `remote hook` is a custom JavaScript function that runs before or after the LoopBack application completes an API operation call.

In this section, you write a function that the LoopBack framework runs after an API operation. When a user submits a review for an item, the remote hook takes an average of all review scores, and updates the value in the database.

For more information about remote hooks, see the LoopBack documentation:

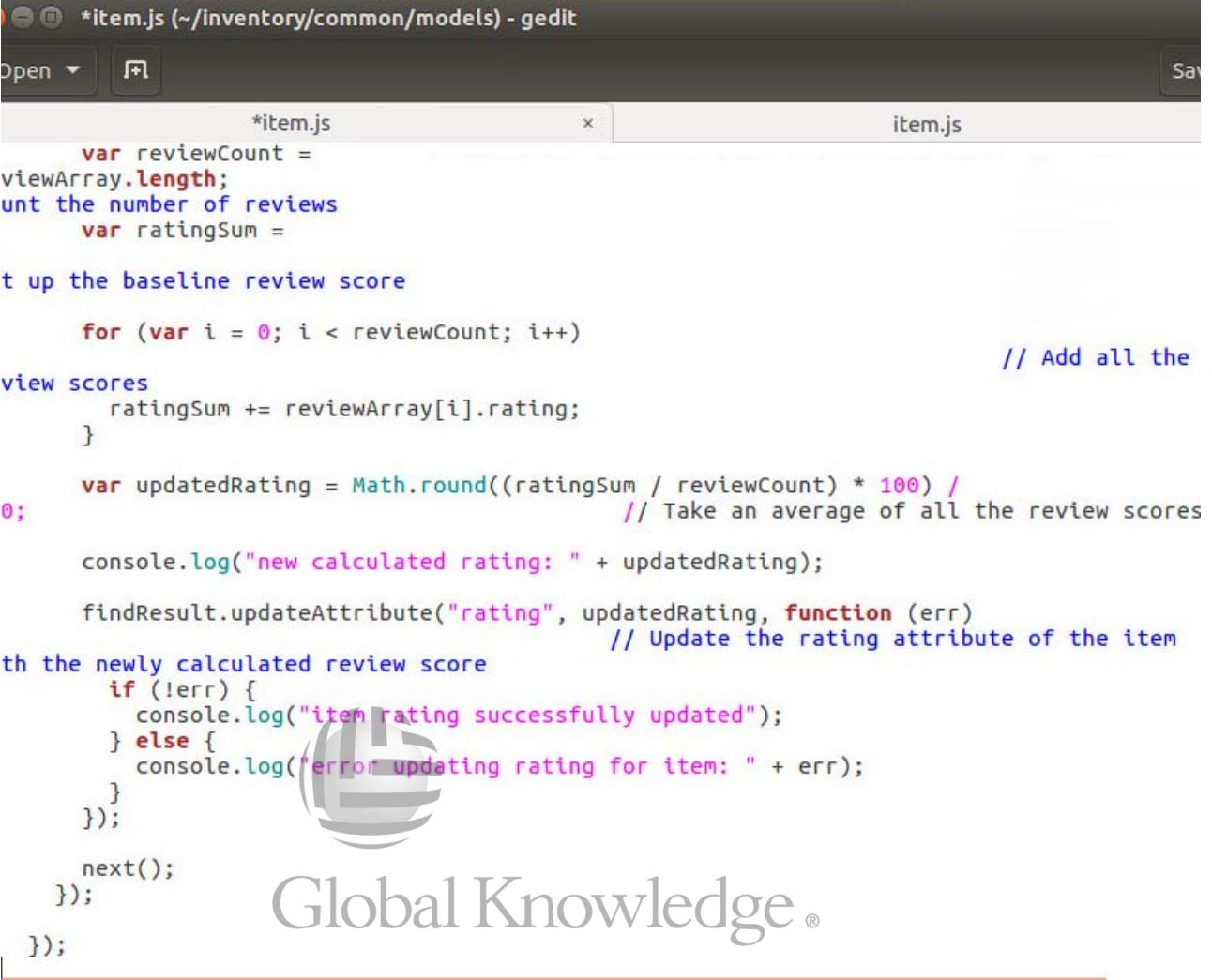
<http://loopback.io/doc/en/lb3/Remote-hooks.html>

- 1. Open a terminal window.
  - a. Confirm that the current directory is in the inventory application.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- 2. Open the `common/models/item.js` model script file with an editor.
  - a. `gedit common/models/item.js`  
You see the script that you updated earlier.
- 3. Copy the remote hook implementation from the solution file.
  - a. Open another text editor from the text editor menu.  
**File > Open**
  - b. Go to the `/home/localuser/lab-files/remote/` folder.
  - c. Select the `item.js` script.  
Click **Open**.
  - d. In the menu bar, click **Edit > Select All** to highlight the contents of the `item.js` script.
  - e. Press **Ctrl+C** to copy the contents of the `item.js` script into the system clipboard.
  - f. In the original text editor, remove the contents of the `item.js` script file.
  - g. Click **Edit > Paste** from the menu bar in the Atom editor.

h. Click **Save**.



```

*item.js (~/inventory/common/models) - gedit
Open ▾ Save
*item.js item.js
var reviewCount =
viewArray.length;
unt the number of reviews
var ratingSum =

t up the baseline review score

for (var i = 0; i < reviewCount; i++) // Add all the
view scores
ratingSum += reviewArray[i].rating;
}

var updatedRating = Math.round((ratingSum / reviewCount) * 100) /
0; // Take an average of all the review scores

console.log("new calculated rating: " + updatedRating);

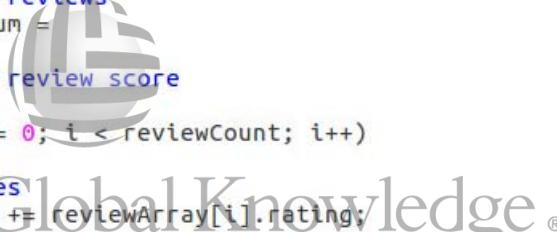
findResult.updateAttribute("rating", updatedRating, function (err) // Update the rating attribute of the item
th the newly calculated review score
if (!err) {
 console.log("item rating successfully updated");
} else {
 console.log("error updating rating for item: " + err);
});
next();
});
});

```

Global Knowledge®

4. Examine the remote hook implementation in the Atom editor.

a. Review the contents of the `item.js` script file.



```

item.js (~/inventory/common/models) - gedit
Save

Item.afterRemote('prototype.__create__reviews', function (ctx,
remoteMethodOutput, next) {
 // Set up a function to run
 // after a review is created
 var itemId =
remoteMethodOutput.itemId;
Get the id of the item that the review was just created for

 console.log("calculating new rating for item: " + itemId);

 var searchQuery = {include: {relation:
'reviews'}};
 // Set
up the search query to find all the existing reviews for the item

 Item.findById(itemId, searchQuery, function findItemReviewRatings(err,
findResult) {
 // Run the search and save the results
 // to a variable called findResult
 var reviewArray = findResult.reviews
();
 Store each of the reviews in an array
 var reviewCount =
reviewArray.length;
 Count the number of reviews
 var ratingSum =
0;
 Set up the baseline review score

 for (var i = 0; i < reviewCount; i++)
 // Add
 all the review scores
 ratingSum += reviewArray[i].rating;
 }

 var updatedRating = Math.round((ratingSum / reviewCount) * 100) /
100;
 // Take an average of all the
 }
}

```



## Information

A **remote hook** declaration has two components: the operation on the model, and script code that LoopBack runs when the user calls the model operation.

In this example, the `Item.afterRemote` listens to the `prototype.__create__reviews` event. After the LoopBack framework creates a `review` model instance, the framework triggers the anonymous function.

```

Item.afterRemote(
 'prototype.__create__reviews',
 function (ctx, remoteMethodOutput, next) {
...
}

```

The anonymous function holds the logic for the remote hook. The function accepts three input variables:

- The context variable `ctx` captures the environment settings of the LoopBack environment. It holds the original request message for the create review API operation.
  - The `remoteMethodOutput` variable captures the response from the intercepted API call. In this example, it is the create review API operation response.
  - When your remote method finishes its work, it starts the `next` callback handler to pass control back to the LoopBack framework. You must call the `next` function; otherwise, the LoopBack API application does not know whether the event handler successfully completed its work.
- 



- \_\_ b. Examine the code within the anonymous function:

```

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;
 console.log("new calculated rating: " + updatedRating);

 findResult.updateAttribute(
 "rating",
 updatedRating,
 function (err) {
 if (!err) {
 console.log("item rating successfully updated");
 } else {
 console.log("error updating rating for item: " + err);
 }
 }
);
 next();
 });
 });
};

```



## Information

In this example, the `Item.afterRemote('prototype.__create__reviews', ...)` function listens to the create reviews operation. After the operation completes successfully, LoopBack calls the anonymous function that is stated as the second parameter of the `Item.afterRemote` call.

- \_\_\_ c. Examine the `findById` function call in the remote hook function implementation.

```
var searchQuery = {include: {relation: 'reviews'}};
Item.findById(
 itemId,
 searchQuery,
 function findItemReviewRatings(err, findResult) {
 ...
});
```



## Information

The `Item.findById` function returns the item model with an identifier that matches the value of `itemId`. The search query includes the relationship that is named `reviews`. The purpose of the search query is to capture a list of review models that correspond to the item in question.

The `findById` function returns the search results to the `findResult` input parameter.

- \_\_\_ d. Examine the first half of the `findItemReviewRatings` function.

```
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;
```



## Information

The `findItemReviewRatings` function processes the search results from the `Item.findById` function.

In the first half of this function, the `findResults.reviews()` call returns an array of review model object- that belongs to the item model. In other words, the function returns all reviews for the item in question. The section beginning with the `for` loop calculates the average review rating for this item.

- \_\_ e. Review the remaining half of the **findItemReviewRatings** function.

```
findResult.updateAttribute(
 "rating",
 updatedRating,
 function (err) {
 if (!err) {
 console.log("item rating successfully updated");
 } else {
 console.log("error updating rating for item: "+ err);
 }
 }
};
next();
```

---



### Information

In the last step, you calculated and saved the average rating value in the `updatedRating` variable. The `findResult.updateAttribute` function updates the `item.rating` property.

The last command in the `findItemReviewRatings` function is the call to the `next` function. This function notifies the LoopBack framework that the remote hook completed its operation. You must add a call to the `next` function at the end of a remote hook.

- 
- \_\_ 5. Close the text editor.



Global Knowledge ®

## 6.5. Test the remote hook

To test the remote hook, create a review for an item in the inventory application. Use the LoopBack API Explorer to generate a sample review for the item record with an **id** value of 1.

The test operation also triggers the after save operation hook. The log function records the item record that is modified in the application log.

- \_\_\_ 1. Start the application from the terminal.
  - \_\_\_ a. Confirm that the current directory is in the inventory application.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

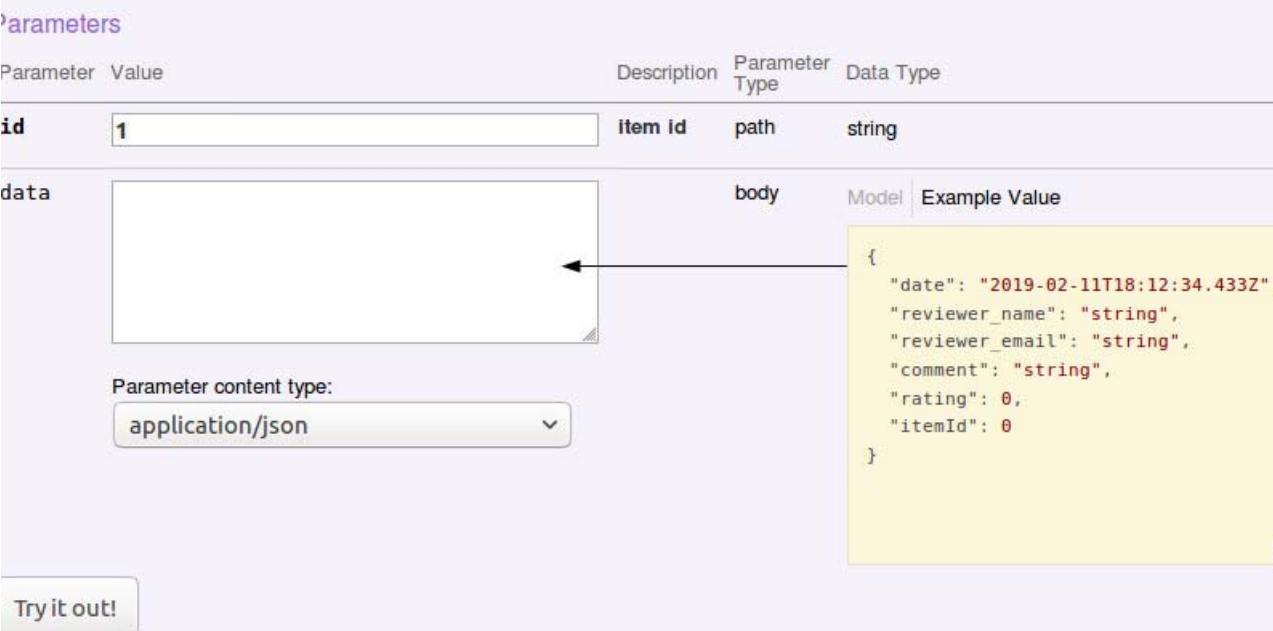
- \_\_\_ b. Start the Loopback application.

```
$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .
```

```
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- \_\_\_ c. Open the API Explorer in the browser. Type:  
<http://localhost:3000/explorer>
  - \_\_\_ d. The LoopBack API Explorer opens in the browser window.
  - \_\_\_ e. In the LoopBack API Explorer, click **List of Operations**.
- \_\_\_ 2. Create a review for the item record with an identifier value of 1.
    - \_\_\_ a. In the API Explorer, select the `post /item/{id}/reviews` operation.

- \_\_\_ b. In the parameters area, set the **item id** to **1**. This parameter is the identifier for the item record that has a relationship to the review.  
 Then, click in the example area to copy the sample body content to the data area.



The screenshot shows the 'Parameters' section of the IBM Cloud Functions interface. A table lists parameters with their values, descriptions, parameter types, and data types. The 'id' parameter is set to '1'. The 'data' parameter is set to a JSON object. To the right of the 'data' row is a yellow box containing the JSON sample body. Below the table is a dropdown menu for 'Parameter content type' set to 'application/json'. At the bottom left is a 'Try it out!' button.

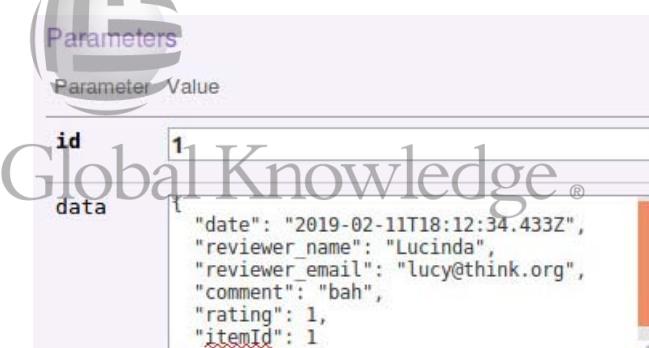
| Parameter   | Value         | Description | Parameter Type | Data Type     |
|-------------|---------------|-------------|----------------|---------------|
| <b>id</b>   | <b>1</b>      | Item id     | path           | string        |
| <b>data</b> | [JSON Object] | body        | Model          | Example Value |

```
{
 "date": "2019-02-11T18:12:34.433Z",
 "reviewer_name": "string",
 "reviewer_email": "string",
 "comment": "string",
 "rating": 0,
 "itemId": 0
}
```

Parameter content type:  
application/json

Try it out!

- \_\_\_ c. Change the sample data to values of your choosing.



The screenshot shows the 'Parameters' section with the 'id' value changed to '1'. The 'data' value is a JSON object with modified fields: 'reviewer\_name' is 'Lucinda', 'reviewer\_email' is 'lucy@think.org', 'comment' is 'bah', 'rating' is 1, and 'itemId' is 1. An orange eraser icon is visible next to the 'data' input field.

| Parameter   | Value         |
|-------------|---------------|
| <b>id</b>   | <b>1</b>      |
| <b>data</b> | [JSON Object] |

```
{
 "date": "2019-02-11T18:12:34.433Z",
 "reviewer_name": "Lucinda",
 "reviewer_email": "lucy@think.org",
 "comment": "bah",
 "rating": 1,
 "itemId": 1
}
```

- \_\_\_ d. Click **Try it out**.  
 \_\_\_ e. Confirm that the operation created the item review.



The screenshot shows the 'Response Body' section displaying the JSON response of the created item review. The response includes the date, reviewer name, reviewer email, comment, rating, ID, and item ID.

```
{
 "date": "2019-02-11T18:12:34.433Z",
 "reviewer_name": "Lucinda",
 "reviewer_email": "lucy@think.org",
 "comment": "bah",
 "rating": 1,
 "id": "5c61bfb3da15ed85d286794f",
 "itemId": 1
}
```

- \_\_\_ f. Confirm that the **after save** operation hook logged the action in the application log.

```
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
Accessed item matching {"id":1}
calculating new rating for item: 1
Accessed item matching {"id":1}
new calculated rating: 3.5
Saved item#1
item rating successfully updated
```



### Information

When you call the `POST /item/1/reviews` operation, the **remote hook** intercepts the call and looks up the reviews that belong to the item. The function calculates the average review rating, and it updates the `Item.rating` property with the value.

The **after save** operation hook also intercepts the update operation on the item model object. It wrote the entry “Saved item#1” in the application log.

- 
- \_\_\_ 3. Close the Loopback API Explorer in the browser.
  - \_\_\_ 4. Click CTRL-C to stop the inventory application that is running in the terminal.

**End of exercise**

Global Knowledge®

## Exercise review and wrap-up

The first part of the exercise examined the role of operation hooks: event listeners that act on access or modifications to data source records that back the model objects.

The second part of the exercise examined the role of remote hooks: event listeners that act before or after the LoopBack framework calls the implementation for an API operation.



# Exercise 7. Assemble message processing policies

## Estimated time

01:30

## Overview

This exercise explains how to create message processing policies. You define a sequence of policies in the assembly view of the API Manager. You define an API that exposes an existing SOAP service as a REST API. You also define an API that transforms responses from an existing service into a defined message format.

## Objectives

After completing this exercise, you should be able to:

- Import an OpenAPI definition into API Manager
- Test the OpenAPI definition in API Manager
- Create an API with JSON object definitions and paths
- Configure an API to call an existing SOAP service
- Import an existing API definition into the source editor
- Create an assembly with a switch that has a flow for each API operation
- Define a gateway script with an API assembly that saves a variable and calls an external map service
- Test DataPower gateway policies in the API Manager assembly view

## Introduction

Message processing policies are a set of processing actions that you apply to API operation request and response messages. You assemble message processing policies as a sequence of actions in the **assembly view** of the API Manager web application.

The purpose of message-processing policies is to process requests to an API operation at the HTTP message level. You can transform, validate, or process a request message before it reaches the API implementation. You can also add logic constructs that route messages based on the content of the request message.

The API gateway enforces the message processing policies that you define in the API definition file. In effect, the message policies apply to all operations that you defined in the API definition.

The types of message processing policies that you can apply depend on your choice of API gateway type. A number of policies run only on DataPower API Gateways.

In this exercise, you define two more API definitions: *financing* and *logistics*. In the *financing* API, you define a policy that converts REST API operations to SOAP API service requests. In the *logistics* API, you define a sequence of processing policies that extracts data from a remote service.

## Requirements

Before you start this exercise, you must complete the data sources and remote exercises in this course.

You must complete this lab exercise on the remote student image: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



## Exercise instructions

### Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



## 7.1. Import the inventory API definition into API Manager

In this section, you import the OpenAPI definition in your `inventory` application into API Manager. When the API definition is in API Manager, you can manage how the API is called. You can also publish the API definition so that the API becomes callable on the API Gateway.

- \_\_\_ 5. Generate the API definition file that you import into API Manager.

- \_\_\_ a. Ensure that you are in the `inventory` directory.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- \_\_\_ b. In the terminal, type the commands to create the API definition file.

```
$ apic lb export-api-def > inventory.yaml
```

The OpenAPI definition that is named `inventory.yaml` is created in the current directory.

- \_\_\_ 6. Open the API Manager web application.

- \_\_\_ a. Open a browser window.

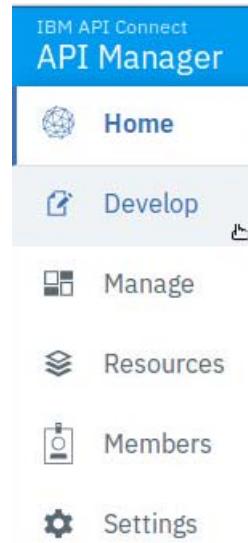
Then, type `https://manager.think.ibm.`

- \_\_\_ b. Sign in to API Manager. Type the credentials:

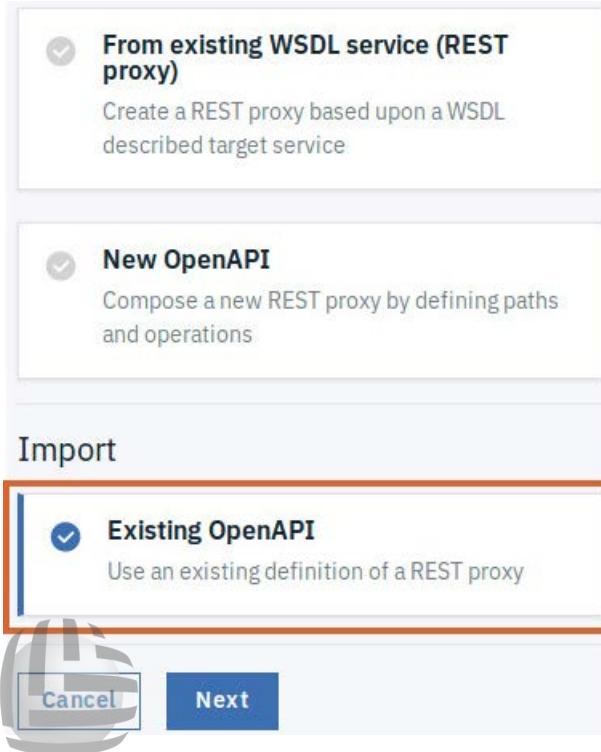
- User name: ThinkOwner
  - Password: Passw0rd!

Click **Sign in**.  
You are signed in to API Manager.

- \_\_\_ 7. Click the tile Develop APIs and Products, or select the Develop option from the side menu.



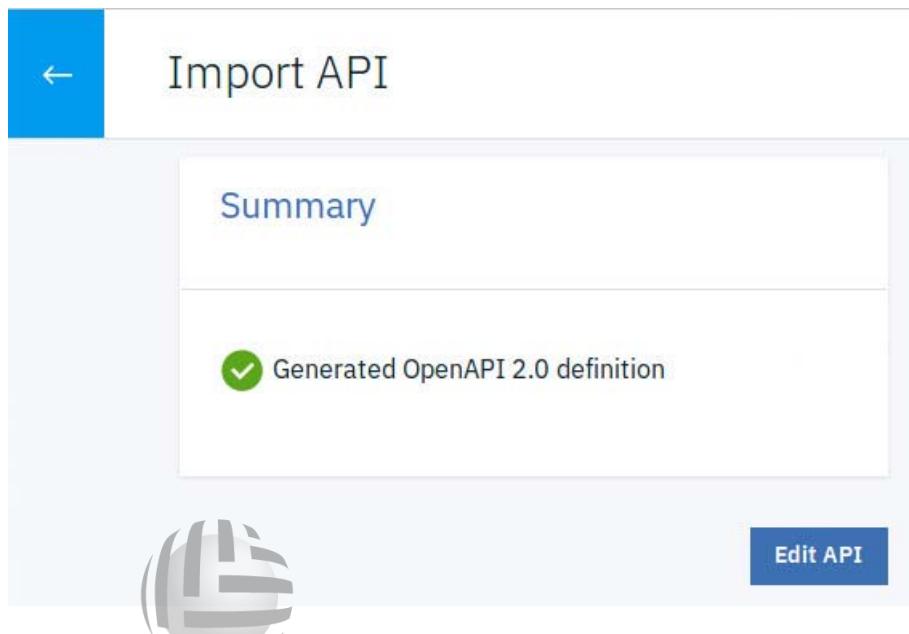
- \_\_ 8. Create an API definition by importing the `Inventory.yaml` file into API Manager
  - \_\_ a. In the API Manager develop page, click **ADD**.  
Then, select **API**.
  - \_\_ b. On the Add API page, select **Existing OpenAPI**.



- The selector is highlighted.
- \_\_ c. Click **Next**.
  - \_\_ d. In the Import from file page, click **Browse**.

The screenshot shows the 'Import from file' dialog. It has a large dashed box in the center for dragging and dropping files, with an upward arrow icon above it. Below the box, the text 'Drag & Drop your file here, or' is displayed. A 'Browse' button is located below this text. At the bottom of the dialog are 'Cancel' and 'Next' buttons.

- \_\_ e. Select the `inventory.yaml` document from the `/home/localuser/inventory` directory. Then, click **Open**.
- \_\_ f. The YAML is imported and successfully validated.
- \_\_ g. Click **Next**.
- \_\_ h. Leave the Activate API option cleared. Click **Next**.
- \_\_ i. The generated OpenAPI 2.0 definition is created.



- \_\_ j. Click **Edit API**.  
The inventory API is displayed in the design view.
- \_\_ k. Scroll down with API Setup selected. Notice that the base path is set to `/inventory`.

- \_\_ I. Click the **Paths** category.

The screenshot shows the LoopBack API Designer interface. At the top, there are two tabs: "Develop" and "Design". Below them, the title "inventory 1.0.0" is displayed. On the left, a sidebar lists several categories: API Setup, Security Definitions, Security, Paths (which is highlighted with a blue background), Definitions, Properties, Target Services, Categories, and Activity Log. To the right, the main content area is titled "Paths". It contains a table with a single column labeled "NAME" and several rows of API paths: "/items/{id}/reviews/{fk}", "/items/{id}/reviews", "/items/{id}/reviews/count", "/items", "/items/replaceOrCreate", and "/items/upsertWithWhere".

Notice that the paths were created when the OpenAPI definition was generated. These paths correspond to the paths that you saw earlier in the LoopBack Explorer.

- \_\_ m. Click the **Properties** category in the design view. Then, click **Add**.

The screenshot shows the LoopBack API Designer interface. At the top, there are two tabs: "Develop" and "Design". Below them, the title "Global Knowledge" is displayed. On the left, a sidebar lists several categories: API Setup, Security Definitions, Security, Paths, Definitions, and Properties (which is highlighted with a blue background). To the right, the main content area is titled "Properties". It contains a table with columns: PROPERTY NAME, ENCODED, and DESCRIPTION. There is one row visible: "target-url" (PROPERTY NAME), "false" (ENCODED), and "The URL of the target service" (DESCRIPTION). A blue "Add" button is located at the top right of the properties table.

- \_\_\_ n. Type the following details for the property:
- Name: **app-server**
  - Default value: **http://platform.think.ibm:3000**
  - Description: **Host where the inventory application runs**

## Create Property

### Name

app-server

### Default value (optional)

http://platform.think.ibm:3000

### Description (optional)

Host where the inventory application runs

- \_\_\_ o. Click **Save**.
- \_\_\_ p. The app-server property is added to the list of properties for the inventory API.

| Properties    |         |                                           |
|---------------|---------|-------------------------------------------|
| PROPERTY NAME | ENCODED | DESCRIPTION                               |
| target-url    | false   | The URL of the target service             |
| app-server    | false   | Host where the inventory application runs |

- \_\_\_ q. Click **target-url** in the properties list for the inventory application.

\_\_\_ r. Type the following details for the property:

- Name: **target-url**
- Default value: **`$(app-server)${request.path}${request.search}`**
- Description: **The URL of the target service**

## Edit Property

### Name

target-url

### Default value (optional)

`$(app-server)${request.path}${request.search}`

### Description (optional)

The URL of the target service

\_\_\_ s. Click **Save**.



### Information

The default message processing policy is to call the API application that implements the API operation. The target-url API application endpoint consists of three parts:

- The `app-server` is the user-defined property that includes the host name and port number of the application server that listens for API operation requests.
- The `request.path` is the API path, including the base path concatenated with the operation path, for example: `/inventory/items`
- The `request.search` is the query parameters that limit the results.

The inventory application is now configured to be called from the API Gateway. The back-end service is the LoopBack application that runs on the host that is defined by the `app-server` property.

- \_\_\_ 9. Click the Develop option in the navigation menu to return to the develop APIs and Products.

| APIs and Products                                                                                 |            | Add ▾          |
|---------------------------------------------------------------------------------------------------|------------|----------------|
| Title                                                                                             | Type       | Last Modified  |
|  inventory-1.0.0 | API (REST) | 43 minutes ago |

You see that the inventory API is added in API Manager.



## 7.2. Test the inventory API definition by calling it on the gateway

In this section, you test the API definition that you imported. The Assembly view in API Manager has a test feature where you can publish and test the API.



### Information

The OpenAPI 2.0 definitions for an API, and its implementation are two separate entities. You define the APIs and their message processing policies in API Manager. These definitions are stored internally on API Manager and are managed from the Develop page. The inventory API has a local implementation of a LoopBack application that can be activated by issuing the `npm start` command from the `/home/localuser/inventory` directory on the student image. The API implementation is separate from the message processing policies insofar as the implementation is a target URL specified in the properties of a message policy.

- 1. Open a terminal window and start the LoopBack “back-end” application.

- a. Navigate to the inventory application directory.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- b. Start the Loopback application.

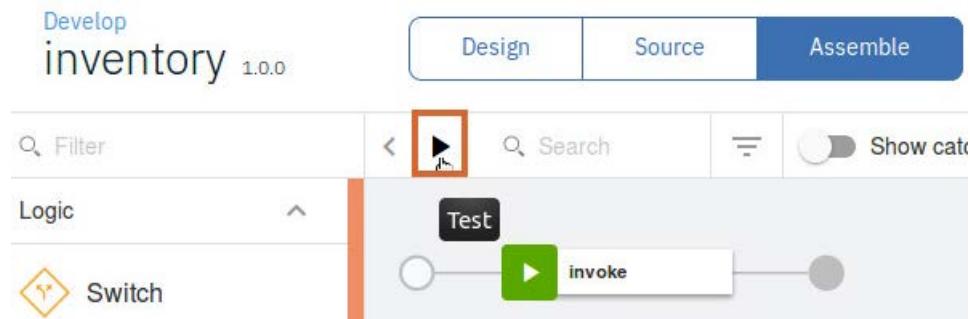
```
$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .
```

Web server listening at: `http://localhost:3000`  
 Browse your REST API at `http://localhost:3000/explorer`

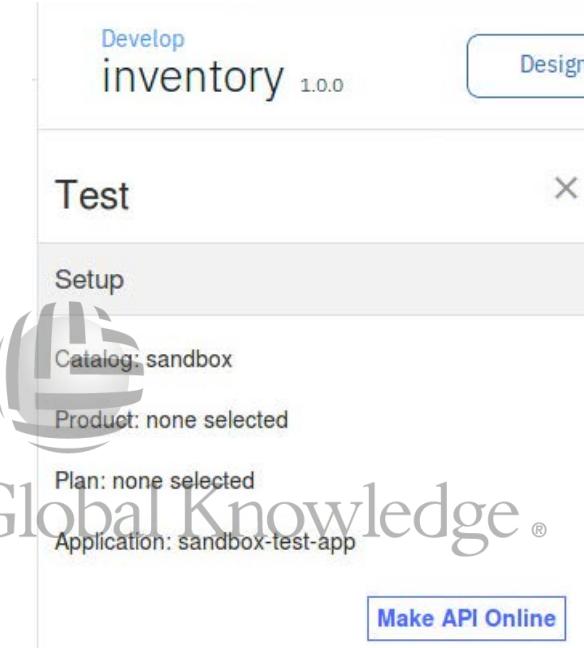
- 2. Open the API definition for the inventory API.

- a. From the API Manager Develop page, click `inventory-1.0.0`.  
 The API opens in the Design view
  - b. Click the **Assemble** tab.

- \_\_\_ 3. Open the test feature for the assembly.
  - \_\_\_ a. In the Assemble page, click the test icon.

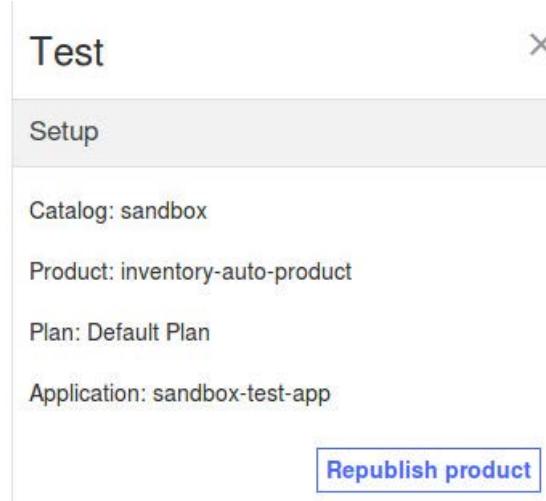


- \_\_\_ b. The test dialog opens.

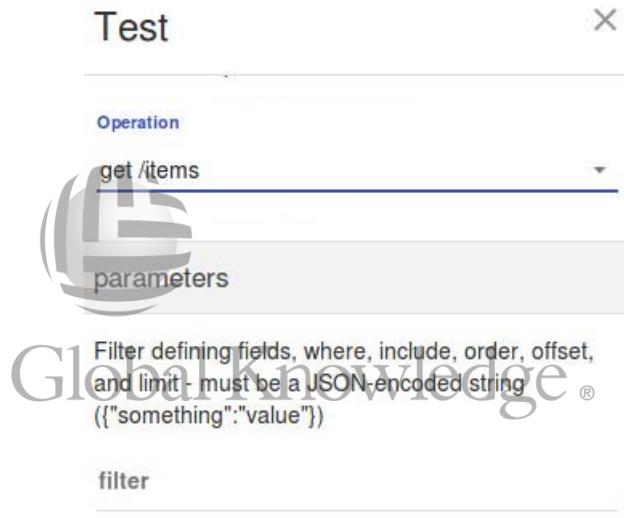


Click **Make API Online**.

- \_\_\_ c. A Product and Plan are now added to the inventory API.

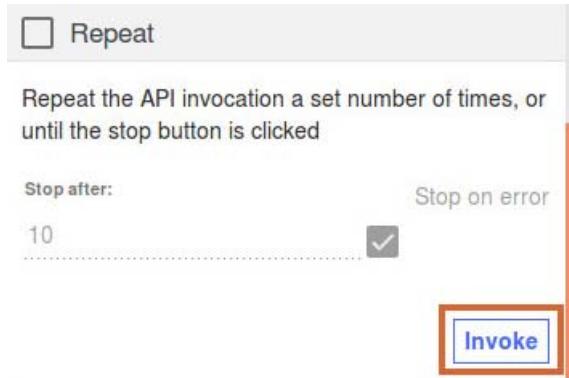


- \_\_\_ d. Scroll down to choose and operation to invoke.



Select `get /items` from the operation drop-down.

- \_\_\_ e. Scroll down, then click **Invoke**.



- \_\_\_ f. The test returns a response with status code 200 OK.

The screenshot shows a 'Test' dialog window with the following details:

- Response**:
  - Status code: 200 OK
  - Response time: 642ms
  - Headers:  
apim-debug-trans-id: 426530149-Landlord-apiconnect-8d11a4fd-e40a-44a1-a073-65556c19c755  
content-type: application/json; charset=utf-8  
x-ratelimit-limit: name=default,100;  
x-ratelimit-remaining: name=default,99;
  - Body:  
[  
 {  
 "name": "Dayton Meat Chopper",

The results are displayed in the body area of the response message in JSON format.  
You successfully called the LoopBack “back-end” API operation from the API gateway.

- \_\_\_ 4. Run the `get /items/count` operation.
- \_\_\_ a. Scroll up in the test area. Select `get/items/count` in the operation area.

b. Click **Invoke**.

The count of the inventory items is displayed.

| Response       |                                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status code:   | 200 OK                                                                                                                                                                                                                      |
| Response time: | 271ms                                                                                                                                                                                                                       |
| Headers:       |                                                                                                                                                                                                                             |
|                | apim-debug-trans-id: 426530149-Landlord-apiconnect-5accf6e1-5afc-494d-948f-65556c19abe3<br>content-type: application/json; charset=utf-8<br>x-ratelimit-limit: name=default,100;<br>x-ratelimit-remaining: name=default,98; |
| Body:          | {<br>"count": 12<br>}                                                                                                                                                                                                       |

c. Close the test window.

d. Type **CTRL-C** to stop the inventory application in the terminal window.

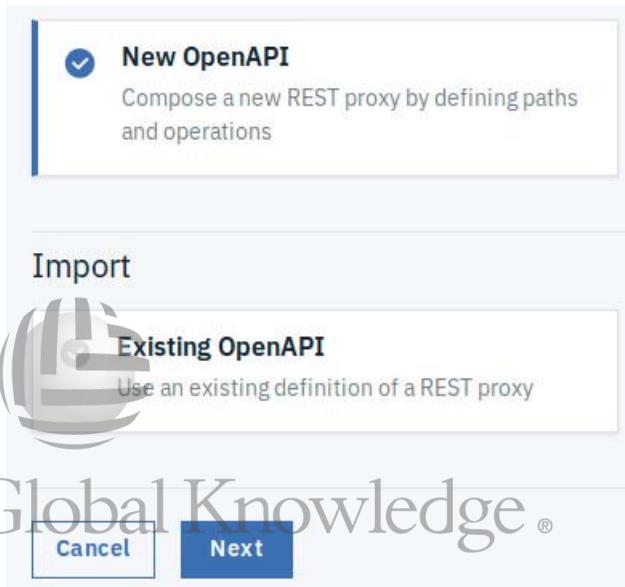
e. You called the inventory application from the API gateway.

Global Knowledge ®

## 7.3. Create the financing API definition

In this section, you define a second API definition that runs along with the `inventory` application: the `financing` API. The financing API calculates monthly payments for items that are selected from the inventory API. Configure the financing API to accept HTTPS connections from API clients. The application type for the request and response message body defaults to: `application/json`

- 1. Create an API definition named `financing`.
  - a. Click the Develop option from the navigation menu.
  - b. In the API Manager develop page, click **Add**. Then, select **API**.
  - c. On the Add API page, select **New OpenAPI**.

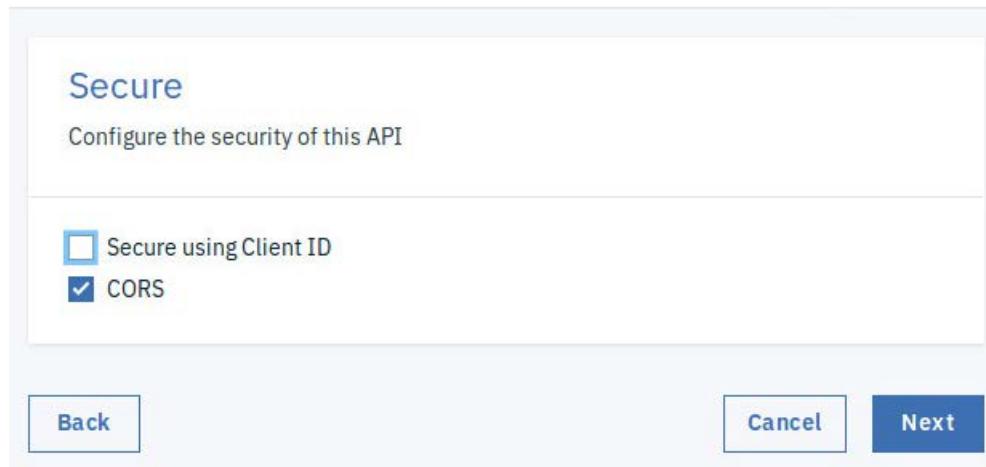


Click **Next**.

- d. Type the following details for the `financing` API definition:
    - Title: `financing`
    - Name: `financing`
    - Version: `1.0.0`
    - Base path: `/financing`
- Click **Next**.

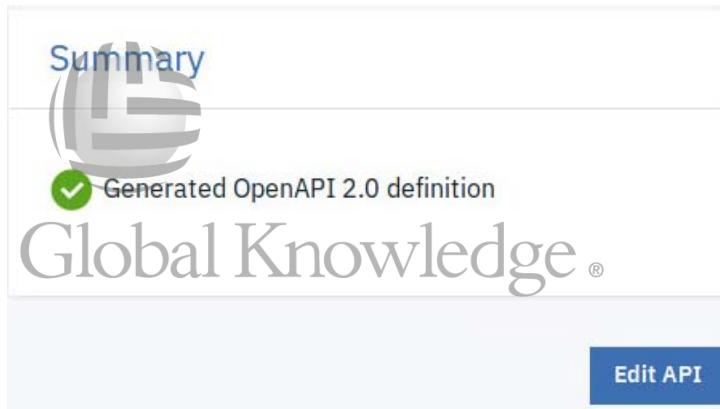
- \_\_\_ e. Clear the option “Secure using client ID” on the secure page of the create new OpenAPI definition.

## Create New OpenAPI



Click **Next**.

- \_\_\_ f. The new OpenAPI definition is generated.



Click **Edit API**.

- \_\_\_ 2. Review the financing API definition.  
\_\_\_ a. In the Design view for the financing API definition, the **API Setup** category, is selected. With the API Setup category selected, scroll down on the page to review the settings.

- \_\_\_ b. In the schemes area on the page, confirm that the scheme is set to https.

## Schemes

Protocols that can be used to invoke the API (only https is supported for gateway enforced APIs)

- HTTP
- HTTPS
- WS
- WSS

The https protocol is the only supported protocol for gateway enforced APIs.

- \_\_\_ c. In the consumes and produces sections, you can leave the defaults (none selected).  
 \_\_\_ d. Scroll to the bottom of the page in the API Setup category. The DataPower API Gateway is selected.

The screenshot shows the 'API Setup' interface. On the left, a sidebar lists options: Security Definitions, Security, Paths, Definitions, Properties, Target Services, Categories, and Activity Log. The 'API Setup' tab is selected. In the main pane, under 'Gateway Type', it says 'Select the gateway type for this API'. Two options are shown: 'DataPower Gateway (v5 compatible)' (unchecked) and 'DataPower API Gateway' (checked). A yellow warning box at the bottom right contains the text: 'The selected gateway type will render the following policies in your assembly as invalid. You will need to delete these policies before you can run this API.' It has a close button 'x' and two buttons: 'Cancel' and 'Save'.

Notice that the gateway type defaults to the DataPower API Gateway for this catalog.



### Note

In the last step, you defined an API named `financing` that accepts API requests at the `/financing` base path. The API accepts https requests with a message body payload.

In the next section, you define a JSON data model named `paymentAmount`. The API operations in the financing API use the `paymentAmount` object as an input parameter.

- \_\_\_ 3. Create an object definition named `paymentAmount`.
- \_\_\_ a. In the financing API, select the **Definitions** category.

The screenshot shows the 'Definitions' page of the Assemble interface. The top navigation bar includes 'Develop', 'financing 1.0.0', 'Design', 'Source', and 'Assemble'. The left sidebar has links for 'API Setup', 'Security Definitions', 'Security', 'Paths', 'Definitions' (which is highlighted), and 'Properties'. The main content area is titled 'Definitions' with a sub-instruction: 'API request and response payload structures are composed using OpenAPI schema definitions.' A large 'Add' button is on the right. Below is a table with columns 'NAME', 'TYPE', and 'DESCRIPTION'. One row is present: NAME is 'paymentAmount', TYPE is 'object', and DESCRIPTION is 'No items found'. There is also a small bee icon.

| NAME          | TYPE   | DESCRIPTION    |
|---------------|--------|----------------|
| paymentAmount | object | No items found |

- \_\_\_ b. Click **Add** to add a definition.
- \_\_\_ c. Change the name of the definition to: `paymentAmount`

The screenshot shows the 'Add Definition' dialog. It has three fields: 'Name' with 'paymentAmount', 'Type' with 'object', and 'Description (optional)' which is empty. A large watermark 'Global Knowledge ®' is visible across the form.

- \_\_\_ d. Click **Add** to add a property.

— e. Type the following property values:

- Property name: `paymentAmount`
- Type: `float`
- Properties example: `199.99`
- Description: `Monthly payment amount`

| PROPERTIES NAME | PROPERTIES TYPE | PROPERTIES EXAMPLE | PROPERTIES DESCRIPTION |
|-----------------|-----------------|--------------------|------------------------|
| paymentAmount   | float ▾         | 199.99             | Monthly payment amount |

— 4. Click **Save**.

The `paymentAmount` object definition is saved.



## 7.4. Create an API operation in the financing API

In the previous section, you defined a definition object named `paymentAmount`. This object represents the financing payment that the API caller types as an input parameter.

In this section, create an API operation that calculates the financing costs. This operation takes a `paymentAmount` data type as an input parameter.

- \_\_\_ 1. Define an API path that is named: `/calculate`
  - \_\_\_ a. In the financing API definition, select the **paths** section.
  - \_\_\_ b. Click **Add** to create a path.
  - \_\_\_ c. Set the path name to `/calculate`.

The screenshot shows the API definition interface. At the top, there is a 'Path name' field containing '/calculate'. Below it, a 'Path Parameters' table is shown with one row. The row has columns: REQUIRED, NAME, LOCATED\_IN, TYPE, DESCRIPTION, and DELETE. The 'NAME' column contains 'Operations' and the 'LOCATED\_IN' column contains a circular icon with a stylized letter 'G'. To the right of the table is a blue 'Add' button. Below the table, the text 'Global Knowledge.®' is displayed next to a small bee logo. At the bottom of the table, it says 'No items found'.

| REQUIRED | NAME       | LOCATED_IN | TYPE | DESCRIPTION | DELETE     |
|----------|------------|------------|------|-------------|------------|
|          | Operations |            |      |             | <b>Add</b> |



### Information

The base path for the financing API is `/financing`. Combined with the API operation path, the URL path for the calculate resource is `https://<hostname>:<port>/financing/calculate`.

- \_\_\_ 2. Define a GET `/calculate` operation that calculates a finance rate. Pass the amount to finance as a query parameter in the GET `/calculate` operation.
  - \_\_\_ a. In operations area of the `/calculate` path, select the **Add** to add an operation.
  - \_\_\_ b. Select the GET operation. Then, click **Add**.  
The GET operation is added.

- \_\_\_ c. Click **Save**.  
The path and operation are saved.
- \_\_\_ d. Click the ellipses in the /calculate path. Then, click **Edit**.
- \_\_\_ e. Go to the GET operation. Click the ellipses, then click **Edit**.
- \_\_\_ f. Set the **operation Id** to get.financingAmount.

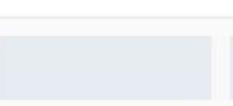


### Information

An operation ID is an HTTP header field that uniquely identifies the API operation.

After an application calls the GET /calculate API operation, the API gateway routes the request through one or more application servers. To track which API operation the client called, the gateway adds an operation ID header in the request message.

- \_\_\_ g. Scroll down in the page. Then, click the **Add** parameter link *three times* to add three parameters.

| Parameters               |      |                                                                                                     |                 |                 |                                                                                       | <b>Add</b> |
|--------------------------|------|-----------------------------------------------------------------------------------------------------|-----------------|-----------------|---------------------------------------------------------------------------------------|------------|
| REQUIRED                 | NAME | LOCATED IN                                                                                          | TYPE            | DESCRIPTION     | DELETE                                                                                |            |
| <input type="checkbox"/> |      |  Choose one... ▾  | Choose one... ▾ |                 |  |            |
| <input type="checkbox"/> |      |  Choose one... ▾ | Choose one... ▾ | Choose one... ▾ |  |            |
| <input type="checkbox"/> |      |  Choose one... ▾ | Choose one... ▾ |                 |  |            |

- h. Set the parameters according to the table of values.

*Table 7. GET /calculate operation parameters*

| Name     | Located in | Description              | Required | Type  |
|----------|------------|--------------------------|----------|-------|
| amount   | Query      | amount to finance        | Yes      | float |
| duration | Query      | length of term in months | Yes      | int32 |
| rate     | Query      | interest rate            | Yes      | float |

| REQUIRED                            | NAME     | LOCATED IN | TYPE   | DESCRIPTION              | DELETE                                                                               |
|-------------------------------------|----------|------------|--------|--------------------------|--------------------------------------------------------------------------------------|
| <input checked="" type="checkbox"/> | amount   | query      | float  | amount to finance        |   |
| <input checked="" type="checkbox"/> | duration | query      | int 32 | length of term in months |   |
| <input checked="" type="checkbox"/> | rate     | query      | float  | interest rate            |  |

- i. Scroll down to the responses section of the GET /calculate API operation. Click **Add**.  
 — j. In the response set the status code to 200, and the description to 200 OK.  
 — k. Set the response message to the paymentAmount model definition that you created earlier.

| Response    |               |             | Add                                                                                   |
|-------------|---------------|-------------|---------------------------------------------------------------------------------------|
| STATUS CODE | SCHEMA        | DESCRIPTION | DELETE                                                                                |
| 200         | paymentAmount | 200 OK      |  |

**Cancel** **Save**

- 3. **Save** the API definition.

The message the API has been updated is displayed and the page reverts to the display the paths.

## 7.5. Set activity logging in the financing API

In this section, you set activity logging for the financing API in the Design view of API Manager.

You use the activity log function to set the content of the API records that are captured on the Gateway for each API that is called on the Gateway service.

- 1. In the Design view with the financing API selected, click the **Activity Log** category.

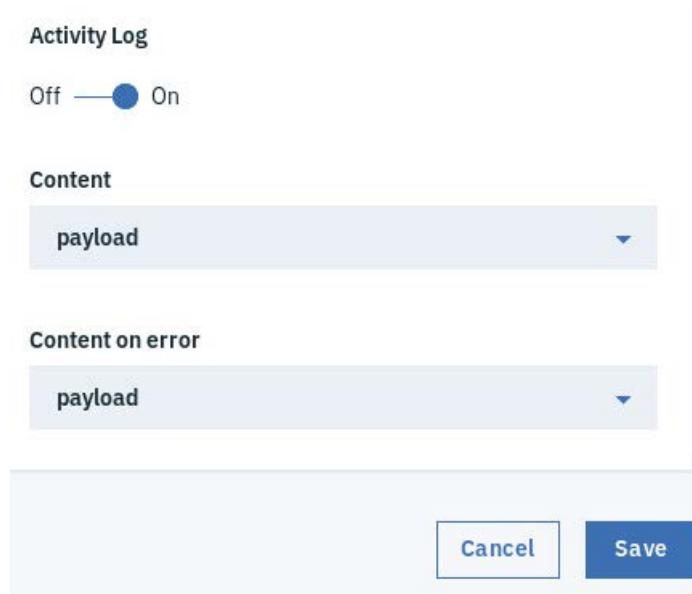
The screenshot shows the API Manager interface. At the top, there's a navigation bar with 'Develop' and 'financing 1.0.0'. Below it, a tab bar has 'Design' selected, followed by 'Source' and 'Assemble'. On the left, a sidebar lists various API setup options: API Setup, Security Definitions, Security, Paths, Definitions, Properties, Target Services, Categories, and Activity Log. The 'Activity Log' option is highlighted with a blue box. The main content area is titled 'Activity Log' with the sub-instruction 'Configure properties of the activity log'. It contains a switch labeled 'Off' with a blue dot and 'On'. Below that is a dropdown menu labeled 'Content' with 'activity' selected. Another dropdown labeled 'Content on error' has 'header' selected. At the bottom right are 'Cancel' and 'Save' buttons.

- 2. Set the activity log for the financing API.

- a. Select the following property values:

- Activity Log: **On**
- Content: **payload**
- Content on error: **payload**

- \_\_\_ b. When you change the content type to payload, a dialog is displayed to enable buffering. Click **Continue**.



- \_\_\_ c. Click **Save**.

The header and body of the request messages are logged when APIs are called successfully and when the call completes with an error code.



## 7.6. Invoke a SOAP web service with a message processing policy

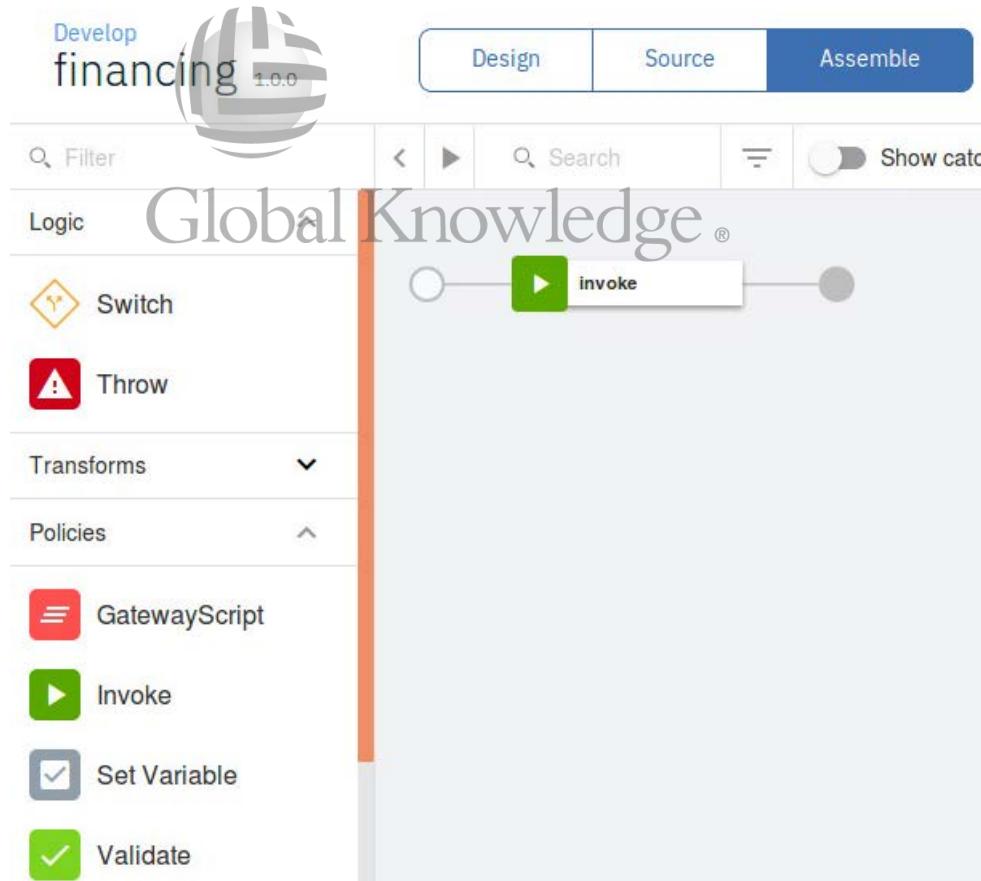
In the last section, you created an API operation that is named GET /calculate in the financing API. The operation takes three query parameters: the amount to finance, the length of the financing term in months, and the interest rate. The operation returns a JSON object of type paymentAmount : the monthly payment cost.

In this section, you assemble a message processing policy that calculates the payment amount. You invoke a remote SOAP web service to calculate the payment amount. You attach the message policy to a REST API operation. In effect, you create a bridge that converts a REST API operation to a SOAP web service invocation. The SOAP web service is a service that runs on the DataPower gateway.

- 1. Open the **Assemble** view for the financing API definition.

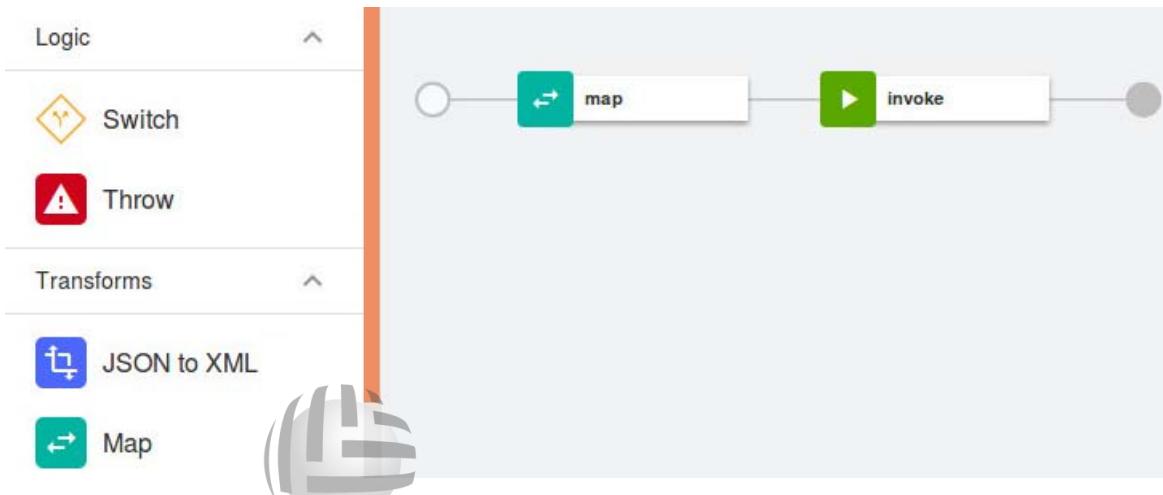


- 2. Since the DataPower Gateway was set as the default gateway for all catalogs, the list of API Gateway-supported policies are displayed in the palette.



An invoke policy is already added to the policy sequence in the free-form area.

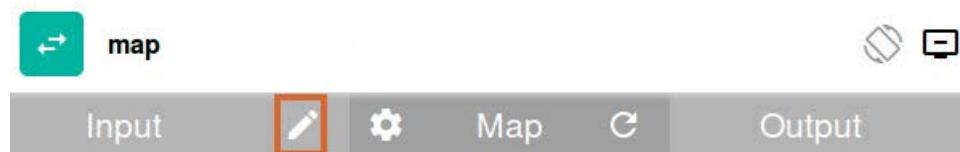
- \_\_\_ 3. You can add logic or policies by dragging them from the left palette onto the canvas. You place the logic or policy in the assembly of components on the canvas. The gateway processes components on the canvas from left to right when the API is called. Policies are grouped in expandable drawers in the palette.
- \_\_\_ 4. Map the input parameters from the GET /calculate API request to the SOAP request message.
  - \_\_\_ a. Expand the Transforms drawer in the palette. Then, select the **map** policy from the policies palette.
  - \_\_\_ b. In the policy pipeline, add a **map** policy between the left, unfilled circle (start) and the **invoke** step.



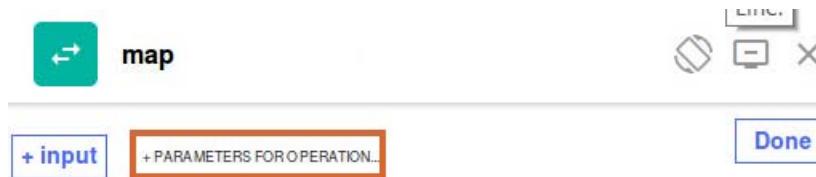
- \_\_\_ c. Open the properties for the **map** policy action. The properties for the map policy open automatically when you drop the policy on the canvas.
- \_\_\_ d. Click the plus sign (+) icon to expand the map properties window.



- \_\_\_ 5. Create three input parameters for the amount, duration, and rate query parameters.
  - \_\_\_ a. In the map properties window, click the edit pencil icon in the **Input** column.



- \_\_\_ b. In the input editor, click **+ parameters for operation**.



- \_\_\_ c. Click the **get.financingAmount** operation to create a set of input parameters based on the request message from the operation.
- \_\_\_ d. Set the definition for the `request.parameters.amount` and `request.parameters.rate` parameters to `float`.

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

- \_\_\_ e. Confirm that the input parameters match the following table.

Table 8. Financing API calculate operation input parameters

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

- \_\_\_ f. Click **Done**.



### Note

To avoid typing errors, you copy the output parameters of the map policy action from a sample schema file.

The `schema_financingSoap.yaml` file contains an OpenAPI definition of the SOAP request message. This definition describes the XML elements and attributes of the SOAP message.

- \_\_\_ 6. Edit the **Output** parameters of the map policy.

- \_\_\_ a. In the map policy editor, click the edit outputs icon from the **Output** column.

| Input            |   |
|------------------|---|
| amount float     | ○ |
| duration integer | ○ |
| rate float       | ○ |

| Output |  |
|--------|--|
|--------|--|

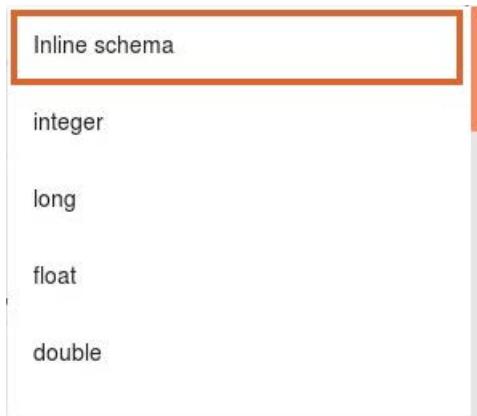
- \_\_\_ b. Click **+ output** to add an output parameter.  
 \_\_\_ c. Leave the context variable to `message.body`.  
 \_\_\_ d. Set the Content type to `application/xml`.

| Context variable          | Name                |
|---------------------------|---------------------|
| <code>message.body</code> | <code>output</code> |

| Content type                 | Definition*         |
|------------------------------|---------------------|
| <code>application/xml</code> | <code>object</code> |

**+ output**      + OUTP UTS FOR OPERATION...      **Done**

- \_\_\_ e. In the **Definition** field, scroll and select `inline schema`.



- \_\_\_ f. Leave the inline schema editor open in your web browser.
- \_\_\_ 7. Copy the output parameters for the `map` policy from the `schema_financingSoap.yaml` file.
- \_\_\_ a. Open the File Manager.
  - \_\_\_ b. Go to the `lab_files/policies` folder.
  - \_\_\_ c. Right-click the `schema_financingSoap.yaml` file. Then, select open with gedit.
  - \_\_\_ d. From the text editor menu, select **Edit > Select All**.
  - \_\_\_ e. Select **Edit >Copy**.
  - \_\_\_ f. Switch back to your web browser. Remove the content in line 1.
  - \_\_\_ g. In the **inline schema** window for the `map output` parameter editor, use CTRL-V to paste the contents of the `schema_financingSoap.yaml` file.

## Provide a schema

The screenshot shows a "Provide a schema" dialog with a code editor. The "Schema as YAML" tab is selected. The code in the editor is:

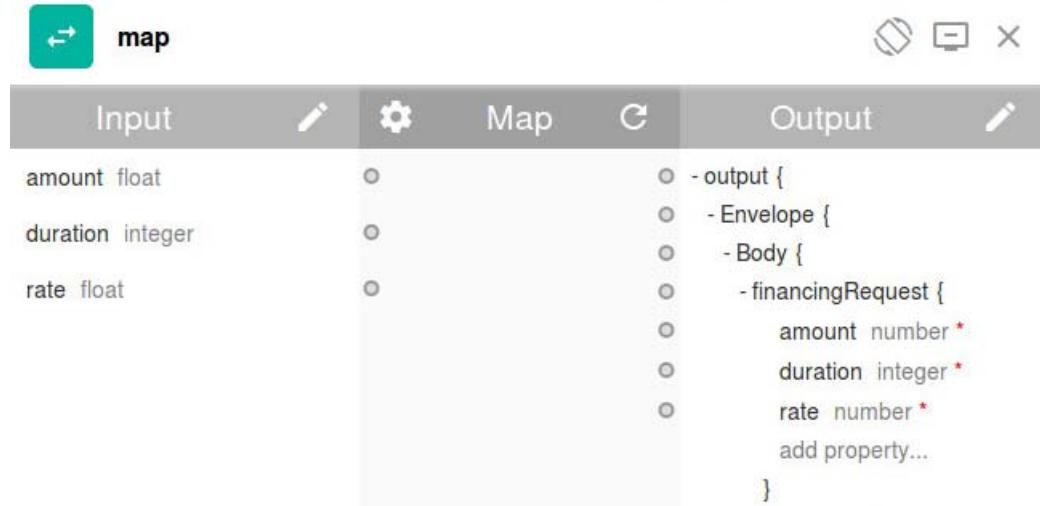
```

28 rate:
29 id: 'http://services.think.ibm/envelope/body/financingRequest/rate'
30 type: number
31 name: rate
32 additionalProperties: false
33 required:
34 - amount
35 - duration
36 - rate
37 name: financingRequest
38 additionalProperties: false
39 required:
40 - financingRequest
41 name: Body
42 additionalProperties: false
43 required:
44 - Body
45 name: Envelope
46 additionalProperties: false
47 required:
48 - Envelope
49 title: output

```

At the bottom right of the dialog are "done" and "cancel" buttons.

- \_\_\_ h. Click **Done** in the inline schema editor.
- \_\_\_ i. Click **Done** in the **output** parameters editor.
- \_\_\_ 8. Review the **map** policy action.



### Information

On the left side, the input to the GET /calculate API operation consists of three query variables: `amount`, `duration`, and `rate`. On the right side, the SOAP web service accepts three parameters in the XML SOAP request message.

Your task is to map the three input parameters to the XML elements of the same name in the output map column. At run time, the API gateway creates a SOAP web service request and copies the values in the message body.

- 
- \_\_\_ 9. Map the `amount`, `duration`, and `rate` API input parameters to the SOAP web service request.
    - \_\_\_ a. Click the node (circle) that follows the `amount` input parameter.
    - \_\_\_ b. Click the node (circle) at the `amount` parameter in the Output column.

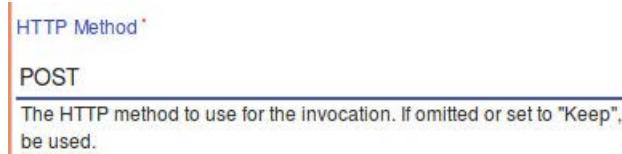
- \_\_\_ c. Repeat the process to connect the `duration` and `rate` parameters to the output parameters of the same name.



- \_\_\_ 10. Close the map policy editor.  
 \_\_\_ 11. Configure the **invoke** policy to call the SOAP web service.  
   \_\_\_ a. Click the **invoke** policy.  
   \_\_\_ b. Change the **URL** field to: <https://services.think.ibm:1443/financing>



- \_\_\_ c. Set the HTTP method to **POST**.



- \_\_\_ d. Close the invoke policy editor.

- \_\_ 12. Configure a **gatewayscript** policy to set the 'content-type' header to: 'application/xml'
- \_\_ a. Drag a **gatewayscript** policy *after* the **invoke** policy in the pipeline.



- \_\_ b. Open the gatewayscript policy editor.
- \_\_ c. Enter the following code:

```
context.message.header.set('Content-Type', 'application/xml');
```



- \_\_ d. Close the gatewayscript editor.
- \_\_ 13. Add a **Parse** policy after the gateway script policy. Close the parse properties window without changes.
- \_\_ 14. Add an **XML to JSON** policy to convert the SOAP service response to a JSON message.
- \_\_ a. Select the **XML to JSON** policy from the transforms palette.
- \_\_ b. Add an **XML to JSON** policy *after* the gatewayscript policy.



Close the XML to JSON properties window.

- \_\_ 15. Save the changes to the **financing** API.



## Information

In this part of this exercise, you created an API definition named financing. The financing API defines one REST API operation: GET /calculate. To call the API operation, you send an HTTP GET request with the following URL:

**`https://<hostname>:<port>/financing/calculate?amount=<amount>&duration=<duration>&rate=<rate>`**

You also defined a message processing policy that transforms the REST API operation into a SOAP web service call.

- The **map** policy copies the amount, duration, and rate query parameters into a SOAP request message. These three input parameters map to XML elements of the same name.
- The **invoke** policy makes a SOAP service request, and captures the response message.
- The **gatewayscript** policy takes the SOAP response message and adds an HTTP header that is named content-type with a value of application/xml.
- The **parse** policy parses the response message according to the content-type set by the gateway script policy
- The **XML to JSON** policy takes the SOAP response message body and converts the value into a JSON message.

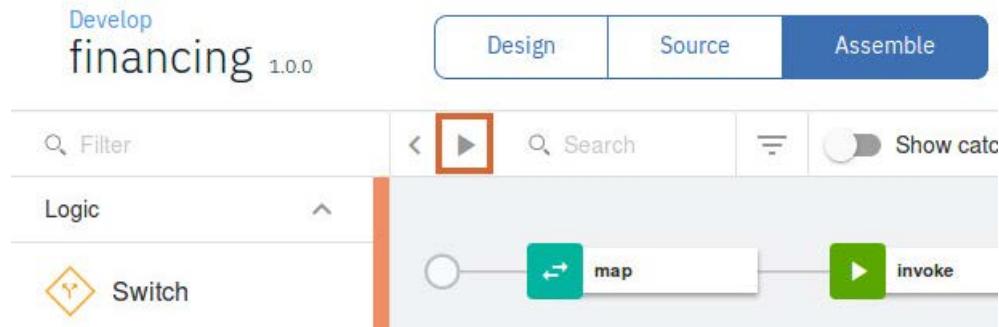


Global Knowledge ®

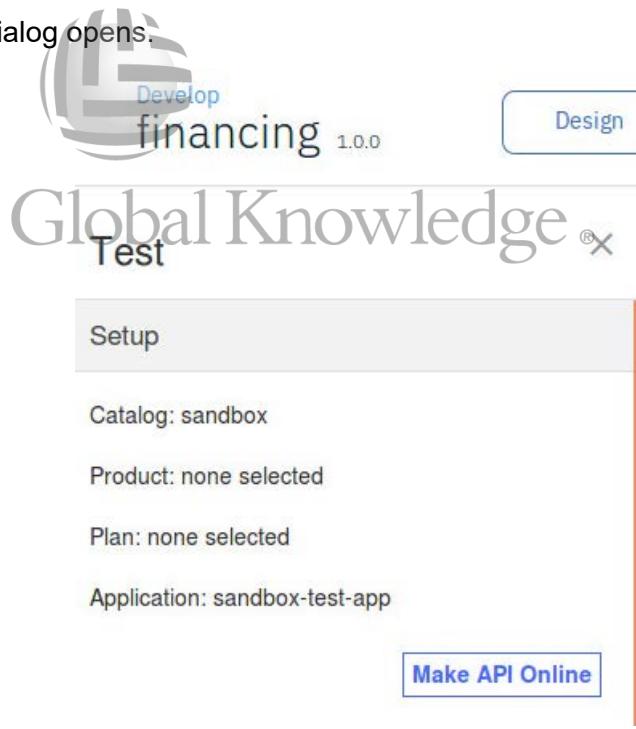
## 7.7. Test the financing APIs by calling it on the gateway

In this section, you auto-publish the API policy assembly in the sandbox environment and test the APIs on the DataPower gateway.

- 1. Test the financing API from API Manager.
  - a. Ensure that the financing-1.0.0 API is open on the Develop page and the Assemble tab is selected.
  - b. Click the test icon in the assembly page.



- c. The test dialog opens.



Click the option **Make API Online**.

The API is added to a Product and a Plan and auto-published.

- d. Scroll down in the test dialog. Then, select the get /calculate operation.
- e. Type the following values for the get /calculate REST API parameters:
  - Amount to finance: 300

- Length in terms of months: **24**
- Interest rate: **0.05**

| parameters                    |      |
|-------------------------------|------|
| amount to finance<br>amount * | 300  |
| <a href="#">Generate</a>      |      |
| term in months<br>duration *  | 24   |
| <a href="#">Generate</a>      |      |
| interest rate<br>rate *       | 0.05 |

- f. Click **Invoke**.
- g. The first time that you call the API on the gateway, you see a message that indicates a lack of CORS support on the target server. Click the link that is provided in the test dialog.



## Global Knowledge Test

| Response                                                                                                                                                                                                            |    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Status code:                                                                                                                                                                                                        | -1 |
| No response received. Causes include a lack of CORS support on the target server, the server being unavailable, or an untrusted certificate being encountered.                                                      |    |
| 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 again.                                                  |    |
| <a href="https://apigw.think.ibm/think/sandbox/financing/calculate?amount=300&amp;duration=24&amp;rate=0.05">https://apigw.think.ibm/think/sandbox/financing/calculate?amount=300&amp;duration=24&amp;rate=0.05</a> |    |

- \_\_ h. Click **Advanced** on the page that displays that your connection is not secure. Then, click **Add Exception**.
- \_\_ i. On the Add Security Exception dialog, accept the default settings. Then, click **Confirm Security Exception**.
- \_\_ j. Confirm that the financing operation returns a 200 OK status code.

**Response**

Status code:

200 OK

Response time:

74ms

- \_\_ k. The response displays the body of the SOAP message. The result is packaged as a JSON object in the HTTP response message.

```

"soapenv:Envelope": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "soapenv:Body": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "ser:financingResult": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "ser:paymentAmount": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "$": "12.51"
 }
 }
 }
}

```

- \_\_ l. You successfully called the financing application from the API gateway.
- \_\_ m. Close the test window.



## Information

If you do not get a successful response from the call, see the section [Appendix , "Testing that the financing back-end service is working on DataPower,"](#) on page B-4



Global Knowledge®

## 7.8. Create the logistics API definition

You define a third API named `logistics`. This API provides two REST APIs:

- The GET `/stores` operation returns the address of a store location based on the postal code (ZIP code) that you provide.
- The GET `/shipping` operation.

In the interest of time, you import a predefined OpenAPI definition file, `logistics_1.0.0.yaml`. You complete the API definition by creating an assembly flows for the GET `/stores` API operation. The GET `/shipping` operation is not completed in this exercise, but as an optional exercise at the end of the course.

- 1. Open the Develop page in the API Manager web application.
- a. Click the Develop icon from the navigation menu. The financing and inventory APIs are displayed.

### Develop

| TITLE           | TYPE       | LAST MODIFIED |
|-----------------|------------|---------------|
| financing-1.0.0 | API (REST) | an hour ago   |
| inventory-1.0.0 | API (REST) | a day ago     |

- b. Select **Add** to define an API. Then, select **API**.
- c. Select the option to import an **Existing OpenAPI**.

Click **Next**.

- d. Click **Browse**. Then, select `logistics_1.0.0.yaml` from the `/home/localuser/lab_files/policies` directory.
- e. Click **Open**.  
The YAML file is successfully validated in API Manager.

- f. Click **Next**.
  - g. Leave the activate API cleared.  
Click **Next**.
  - h. The OpenAPI 2.0 definition is generated.
  - i. Click the return arrow to return to the Develop page in API Manager.
2. The three Open API definitions are displayed in the list of APIs.

| APIs and Products                                                                                 |            |               | Add ▾ |
|---------------------------------------------------------------------------------------------------|------------|---------------|-------|
| TITLE                                                                                             | TYPE       | LAST MODIFIED |       |
|  financing-1.0.0 | API (REST) | an hour ago   |       |
|  inventory-1.0.0 | API (REST) | a day ago     |       |
|  logistics-1.0.0 | API (REST) | 2 minutes ago |       |



### Note

The list of APIs that you see are the OpenAPI 2.0 definitions for 3 APIs. As you saw earlier, the inventory API has a local implementation of a Loopback application that is in the /home/localuser/inventory directory on the student image. The implementation for the financing API is a SOAP service that runs on the Services domain on the DataPower gateway. At this stage, you are not aware of the implementation for the logistics API. The target becomes apparent as you review and update the logistics API. The API implementation is separate from the message processing policies insofar that the implementation is a target URL specified in the properties of a message policy.

## 7.9. Edit the logistics API definition

You change some settings for the logistics API in this part.

- 1. Edit the logistics API.  
Click **logistics-1.0.0** in the list of APIs from the Develop page in the API Manager.  
The API opens in the Design view.
- 2. Change the gateway type to the API Gateway.
  - a. With the API Setup selected, scroll to the bottom of the page.
  - b. Change the gateway type from the v5 compatible gateway to the DataPower API Gateway.

### Gateway Type

Select the gateway type for this API

- DataPower Gateway (v5 compatible)  
 DataPower API Gateway

**⚠** The selected gateway type will render the following policies in your assembly as invalid. You will need **X** to delete these policies before you can run this API.  
 proxy



**Cancel**

**Save**

- c. Click **Save**.  
**Global Knowledge**®  
 You address the warning message later in the assemble view.
- 3. Verify that no security requirements are set.
  - a. Select the **Security** section of the API definition.
  - b. Clear any security requirement, if necessary. Then, click Save.

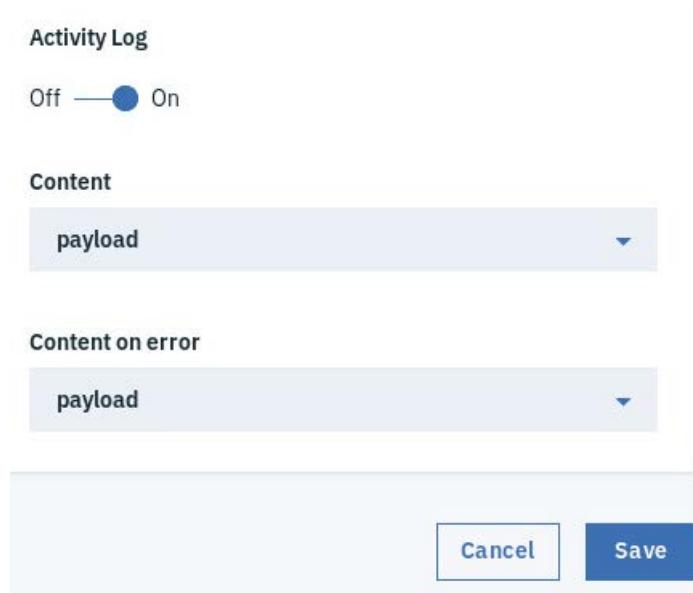
- \_\_\_ 4. In the Design view with the logistics API selected, click the **Activity Log** category.

The screenshot shows the IBM API Connect interface. At the top, there's a header with 'Develop' and 'logistics 1.0.0'. Below it, a navigation bar has tabs for 'Design' (which is selected), 'Source', and 'Assemble'. On the left, a sidebar lists several categories: API Setup, Security Definitions, Security, Paths, Definitions, Properties, Target Services, Categories, and Activity Log. The 'Activity Log' category is highlighted with a blue background. The main content area is titled 'Activity Log' with the sub-instruction 'Configure properties of the activity log'. It contains two sections: 'Activity Log' (with a toggle switch set to 'On') and 'Content' (set to 'activity'). Below that is another section for 'Content on error' (set to 'header'). At the bottom right are 'Cancel' and 'Save' buttons.

- \_\_\_ 5. Set the activity log for the logistics API.

- \_\_\_ a. Click the **Activity Log** category for the logistics API.  
\_\_\_ b. Select the following property values:
- Activity Log: **On**
  - Content: **payload**
  - Content on error: **payload**

- c. When you change the content type to payload, a dialog is displayed to enable buffering. Click **Continue**.



- 6. Click **Save**.



## 7.10. Define the message policies for the GET /stores API operation

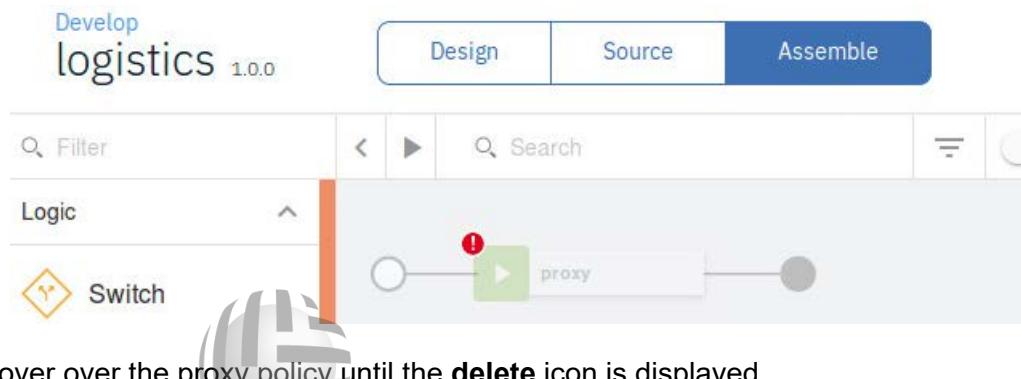
The logistics API defines two API operations: GET /shipping and GET /stores.

Only the GET /stores operation is completed in this exercise.

In this section, you define a sequence of message processing policies for the GET /stores API operation. Call a remote geolocation service to find a store location based on the client's postal code. Add a gateway script policy to format the contents of the API response message.

- 1. In the logistics API, click the **Assemble** tab.

The assembly is displayed. The proxy policy contains an error that was created when the gateway type was changed.



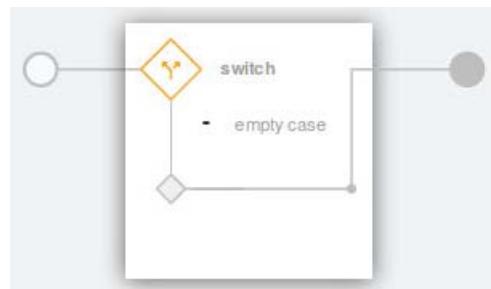
- 2. Hover over the proxy policy until the **delete** icon is displayed.



Click to delete the proxy policy.

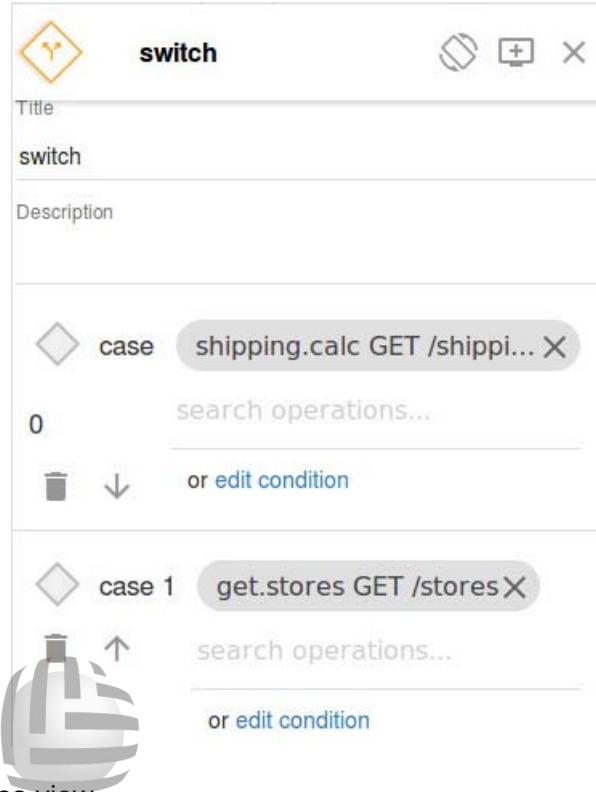
Only the start and end sequences remain.

- 3. Add a Switch policy to distinguish between requests to GET /shipping and GET /stores.
  - a. In the Logic palette, select the **Switch** construct.
  - b. Add the Switch policy in the message sequence between the start and end icon.

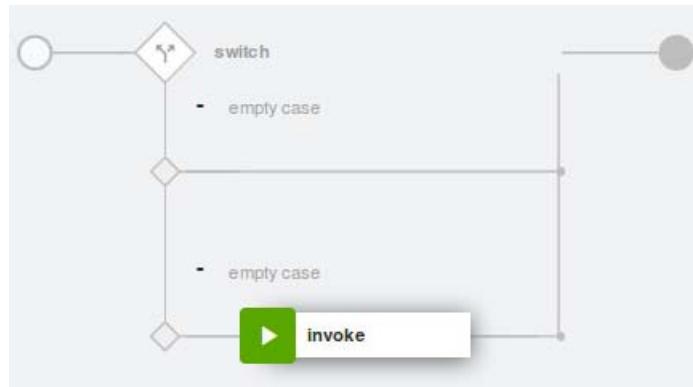


- c. Ensure that the **Switch** policy is open in the Properties view.
- d. Select the **search** operation field in the **case 0** branch.

- e. Select `shipping.calc` GET `shipping`.
- f. Select **+ Case** to create a case for the operation switch.
- g. In **case 1**, select `get.stores` GET `stores` as the operation.



- 4. Close the properties view.
- 5. Add an **invoke** policy to the case 1 condition in the switch.
- a. In the GET /stores case in the switch policy, add an invoke policy.



- b. Open the properties view for the **invoke** policy.
- c. Type the following details for the invoke policy:
  - Title: `invoke_geolocate`

- URL:

<https://nominatim.openstreetmap.org/search?postalcode={zip}&format=json>

The screenshot shows the properties editor for an 'invoke\_geolocate' policy. The title is set to 'invoke\_geolocate'. The URL field contains the value 'openstreetmap.org/search?postalcode={zip}&format=json'. A note below the URL field states: 'The URL to be invoked.'

- Note: The URL that is coded above includes the zip parameter that is surrounded by curly brackets.
- Clear Stop on error.
- Response object variable: **geocode\_response**

The screenshot shows the 'Stop on error' settings for the 'invoke\_geolocate' policy. It includes a note about errors causing the assembly flow to stop, a section for selecting response object variables, and a highlighted 'geocode\_response' variable. A note below the variable states: 'The name of a variable that will be used to store the response data from the request. This can then be referenced in other actions, such as "Map".'



## Information

The Nominatim service geocodes addresses for the Open Street Map project. You provide a part of an address, and the service returns the latitude and longitude coordinates for the location.

- 
- \_\_\_ d. Close the properties editor for the invoke\_geolocate policy.
  - \_\_\_ 6. Add a gatewayscript policy to return a map link for the latitude and longitude coordinates that you received in the invoke policy.
    - \_\_\_ a. Add a **gatewayscript** policy after the invoke policy in the **case 1** operation-script path.
    - \_\_\_ b. Type the following settings in the gatewayscript properties view:

- Title: **format-maps-link**
- \_\_ c. Enter the following script into the policy:

```
// Require API Connect functions
var apim = require('apim');

// Save the geocode service response body to a variable
var mapsApiResponse = apim.getvariable('geocode_response.body');

// Get location attributes from the response message
var location = mapsApiResponse[0];

var storesResponse = {
 "maps_link": 'https://www.openstreetmap.org/#map=16/' +
 location.lat + '/' + location.lon
};

// Save the output
apim.setvariable('message.body', storesResponse);
```



### Note

You can also copy the gateway script source code from the `formatMapsLink.js` script in the `~/lab_files/policies/` directory.

The screenshot shows the Global Knowledge gateway script editor interface. At the top, there's a title bar with the gateway logo and the title "format-maps-link". Below the title bar, there are sections for "Title" and "Description". The "Title" field contains "format-maps-link". The "Description" field is empty. A large text area below contains the script code:

```
1 // Require API Connect functions
2 var apim = require('apim');
3
4 // Save the geocode service response body to a variable
5 var mapsApiResponse = apim.getvariable('geocode_response.body');
6
7 // Get location attributes from the response message
8 var location = mapsApiResponse[0];
9
10 var storesResponse = {
11 "maps_link": 'https://www.openstreetmap.org/#map=16/' +
12 location.lat + '/' + location.lon
13 };
14
15 // Save the output
16 apim.setvariable('message.body', storesResponse);
```

- \_\_ d. Close the gatewayscript editor.

---

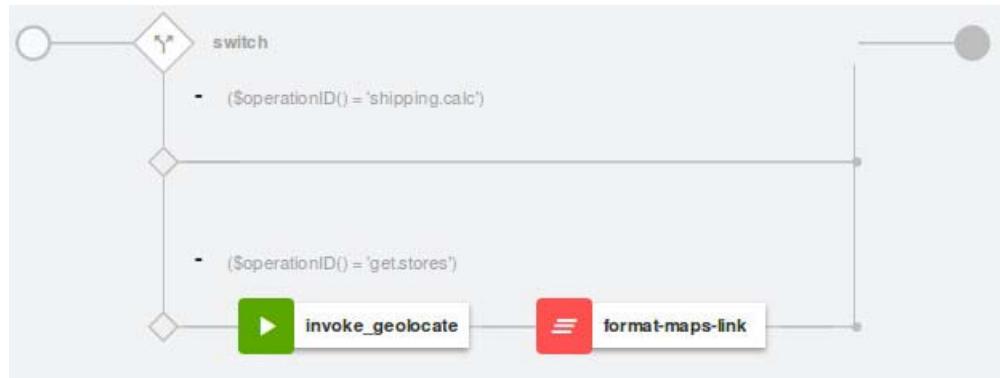
7. Save the logistics API definition.

---



### Information

In the case one branch of the operation switch construct, the GET /stores API operation retrieves the map coordinates based on the US postal code (ZIP code) from the API request. The gateway script policy takes the latitude (location.lat) and longitude (location.lon) values to build an Open Street Map link. The GET /shipping branch is a null operation.

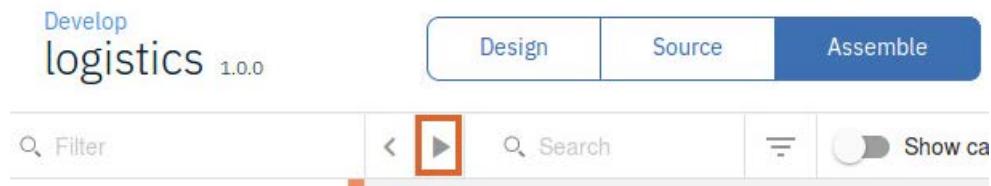


Global Knowledge ®

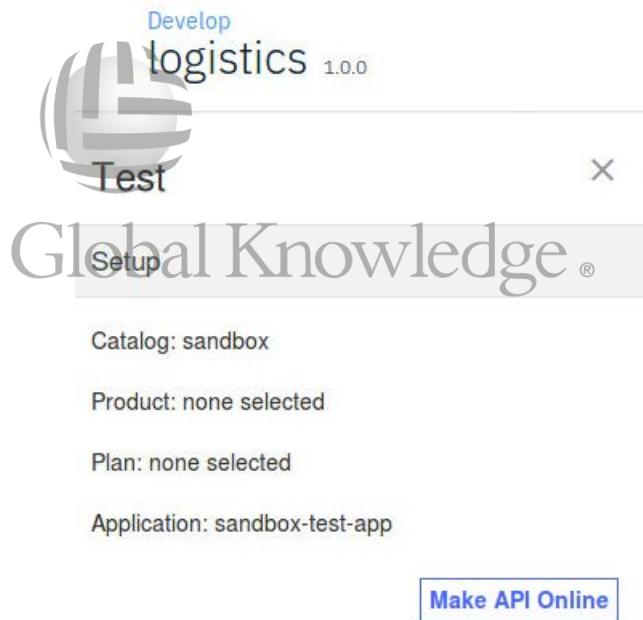
## 7.11. Test the stores API by calling them on the gateway

In this section, you publish the API policy assemblies in the sandbox environment and test the APIs on the DataPower gateway.

- \_\_\_ 1. Test the logistics API from API Manager.
  - \_\_\_ a. From the Develop page, select the `logistics-1.0.0` API.
  - \_\_\_ b. When the API opens in the Design view, click the **Assemble** tab.
  - \_\_\_ c. Click the test icon in the assembly page.



- \_\_\_ d. The test dialog opens.



Click the option **Make API Online**.

The API is added to a Product and a Plan and auto-published.

- \_\_\_ e. Scroll down in the test dialog. Then, select the `get /stores` operation.
- \_\_\_ f. Type the following values for the `get /calculate` REST API parameters:

- zip: 15222

| Operation                      |
|--------------------------------|
| Choose an operation to invoke: |
| Operation<br>get /stores       |
| parameters                     |
| zip *<br>15222                 |

- \_\_\_ g. Click **Invoke**.
- \_\_\_ h. Confirm that the logistics stores operation returns the coordinates of the map as a JSON object in the HTTP response message.

| Response                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  Status code:<br>200 OK<br><br>Response time:<br>601ms<br><br><b>Headers:</b><br>apim-debug-trans-id: 426530149-Landlord-<br>apiconnect-d4036a41-f06b-4a55-880a-<br>65556c196ab7<br>content-type: text/xml<br>x-global-transaction-<br>id: 196c55655c6c8f3500009452<br>x-ratelimit-limit: name=default,100;<br>x-ratelimit-remaining: name=default,97;<br><br><b>Body:</b><br><pre>{   "maps_link": "https://www.openstreet map.org/#map=16/40.44048745/-79.999033 7281745" }</pre> |

The response body returns the map coordinates that correspond to the postcode that you chose.

## End of exercise

## Exercise review and wrap-up

The first part of the exercise examined how to build a REST-to-SOAP bridge at the API gateway. You mapped the parameters from the financing API operation into an SOAP XML message and converted the SOAP response message to a JSON message.

The second part of the exercise examined the logic constructs in the assembly flow. You defined a switch policy to create two subflows: one for the GET /shipping operation, and one for the GET /stores operation.

The GET /shipping operation is stubbed out in the exercise. In the GET /stores operation, you looked up the map coordinates for a store location based on a United States postal code.





Global Knowledge®

# Exercise 8. Declare an OAuth 2.0 provider and security requirement

## Estimated time

01:00

## Overview

In this exercise, you examine two of the three parties in an OAuth 2.0 flow: the OAuth 2.0 provider and the API resource server. You define a Native OAuth provider to authorize access and issue tokens. In the case study application, you declare an OAuth 2.0 security constraint that enforces access control with the OAuth 2.0 provider API.

## Objectives

After completing this exercise, you should be able to:

- Define a Native OAuth provider in the API Manager graphical application
- Configure the client ID and client secret security definition
- Declare and enforce an OAuth 2.0 security definition with the API Manager graphical application

## Introduction

In the case study, the inventory API provides a set of operations that display items in the store and reviews on items. This exercise explains how to secure the inventory API with OAuth 2.0 authorization. You configure a Native OAuth provider: the security server that verifies client identity and access rights to the inventory API operations. The Native OAuth provider also issues and manages access tokens: a time-limited key that allows a client access to API resources.

The Native OAuth provider is a special type of configuration that you define in the API Manager. The Native OAuth provider configuration is added to a catalog. The OAuth provider visibility setting determines which provider organizations can use an OAuth provider to secure APIs. When you publish an API that is secured with the OAuth security definitions to the API gateway, the gateway acts as the security server.

## Requirements

Before you start this exercise, you must complete the data sources, remote, and policies exercises in this course.

You must complete this lab exercise on the student remote development image: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



## Exercise instructions

### Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



Global Knowledge®

## 8.1. Create a Native OAuth Provider

The **Native OAuth Provider** is a preset API configuration: it defines the `/authorize` and `/token` API operations according to the message flow in the OAuth 2.0 specification.

In this section, define a Native OAuth Provider configuration in API Manager. The API Manager application creates the configuration in the resources for the on-premises cloud. You add the configuration to the Sandbox catalog.

- 1. Start the API Manager web application.
  - a. Open a browser window.  
Type `https://manager.think.ibm.`
  - b. Sign in to API Manager. Type the credentials:
    - User name: ThinkOwner
    - Password: Passw0rd!
 Click **Sign in**.  
You are signed in to API Manager.
- 2. Click the **Resources** option from the side menu, or click the Manage Resources tile.
- 3. Add an OAuth Provider API named “oauth”.
  - a. On the Resources page, click the **OAuth Providers** option.  
Click **Add**. Then, select **Native OAuth Provider**.

| TITLE | TYPE                                                |
|-------|-----------------------------------------------------|
|       | Native OAuth provider<br>Third party OAuth provider |

No items found

\_\_ b. Type the following details for the OAuth provider:

- Title: **oauth-provider**
- Name: **oauth-provider**
- Base path: **/oauth20**
- Gateway Type: **DataPower API Gateway**

Title  
oauth-provider

Name  
oauth-provider

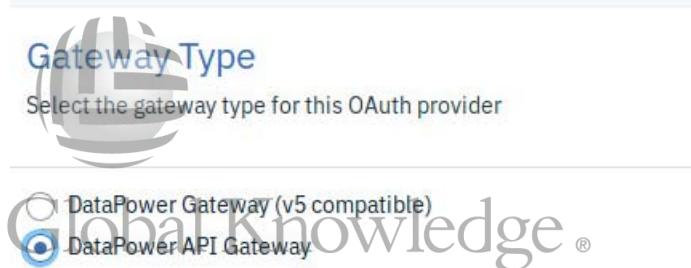
Description (optional)

Base path (optional)  
/oauth20

---

**Gateway Type**  
Select the gateway type for this OAuth provider

DataPower Gateway (v5 compatible)  
 DataPower API Gateway



Click **Next**.



### Important

Remember to change the Base Path field to: **/oauth20** and the gateway to **DataPower API Gateway**.

- \_\_\_ c. Leave the **access code** selected and select the supported grant types field to include **Resource owner password**.

**Authorize path**  
/oauth2/authorize

**Token path**  
/oauth2/token

**Supported grant types**

- Implicit
- Application
- Access code
- Resource owner password

**Supported client types**

- Confidential
- Public

**Back** **Cancel** **Next**

- \_\_\_ d. Leave the default values for the supported client types (confidential). Click **Next**.

- \_\_\_ e. On the Scopes page, type these values:

- Name: **inventory**
- Description: **Access to the inventory API**

| NAME      | DESCRIPTION                 | DELETE |
|-----------|-----------------------------|--------|
| inventory | Access to the inventory API |        |

**Back** **Cancel** **Next**

- \_\_\_ f. Click **Next**.

\_\_ g. On the Identity Extraction page, leave the defaults. The values are:

- Collect credentials using: **Basic Authentication**
- Authenticate application users using: **No LDAP or authentication URL user registries found**
- Authorize application users using: **Authenticated**

The screenshot shows the 'Identity Extraction' configuration page. It has three main sections: 'Collect credentials using', 'Authenticate application users using', and 'Authorize application users using'. In the 'Collect credentials using' section, 'Basic Authentication' is selected. In the 'Authenticate application users using' section, a button labeled 'Create Sample User Registry' is highlighted with a red box. In the 'Authorize application users using' section, 'Authenticated' is selected. The background features a watermark for 'Global Knowledge'.

- h. In the authentication area, click **Create Sample User Registry**. A SampleAuthURL registry is created and added in the authentication drop-down list.

Identity Extraction

---

Collect credentials using

Basic Authentication

---

Authentication

---

Authenticate application users using

SampleAuthURL

---

Authorization

---

Authorize application users using

Authenticated

---

- i. Click **Next**.

Global Knowledge ®

- \_\_j. Review the values on the Summary page.

| NAME      | DESCRIPTION                 |
|-----------|-----------------------------|
| inventory | Access to the inventory API |

## Resource Owner Security

### Collect credentials using

Basic Authentication

### Authenticate application users using

sampleauth

### Authorize application users using

Authenticated

Back

Cancel

Finish

Then, click **Finish**.  
The native OAuth Provider is created.

Global Knowledge ®

- \_\_\_ 4. Examine and edit the definition in the Edit Native OAuth Provider.
- \_\_\_ a. The Info section is displayed. Scroll down and select **Enable debug response headers**.

The screenshot shows the 'Info' tab of a configuration dialog. On the left is a sidebar with links: Info (selected), Configuration, Scopes, User Security, Tokens, Token Management, Introspection, Metadata, OpenID Connect, and API Editor. The main area has fields for Name (set to 'oauth-provider'), Description (optional) (empty), Gateway version (set to '6000'), Base path (set to '/oauth20'), and a checked checkbox for 'Enable debug response headers'. At the bottom are 'Cancel' and 'Save' buttons.

Click **Save**. Global Knowledge ®

- \_\_\_ b. Click the **Configuration** tab.

## Information

The OAuth Provider defines the two API operations according to the OAuth 2.0 specification: /oauth2/authorize, and /oauth2/token.

In the first step, the /oauth2/authorize operation verifies the identity of the client, and determines whether that client has access rights to a certain resource. If it allows the client access to the resource, the authorize operation returns an authorization code.

In the second step, the client sends the authorization code to the /oauth2/token operation. If the authorization code is valid, the token operation exchanges the authorization code for an access token.

When the client application calls an API that is secured according to the OAuth 2.0 scheme, it sends the access token as part of the call.

- \_\_\_ c. Click the return arrow on the page or **Cancel** to exit the edit native oauth provider.

- \_\_\_ 5. Update the URL in the authentication URL user registry.
  - \_\_\_ a. On the Resources page, click **User Registries**.
  - \_\_\_ b. In the list of user registries, click **SampleAuthURL**.

## Resources

| User Registries | User Registries               |                     | Create |
|-----------------|-------------------------------|---------------------|--------|
|                 | TITLE                         | TYPE                |        |
| TLS             | SampleAuthURL                 | Authentication URL  | :      |
| OAuth Providers | Sandbox Catalog User Registry | Local User Registry | :      |

- \_\_\_ c. In the Authentication URL User Registry dialog, type:
  - URL: `https://services.think.ibm:1443/authorize`
- \_\_\_ d. Click **Save**.



### Important

Due to a probable defect in the UI in the installed version of the software, you need to reset the user security setting in the OAuth provider.

- \_\_\_ e. From the Resources page, select OAuth Providers. Then, click **oauth-provider**.
- \_\_\_ f. Click **User Security** on the edit native oauth provider page.

- \_\_\_ g. In the Authentication area, change the value in the authenticate application users from None to **SampleAuthURL**.

The screenshot shows two sections: 'Authentication' and 'Authorization'.  
In the 'Authentication' section, there is a dropdown menu labeled 'Authenticate application users using (optional)' with the value 'SampleAuthURL' selected. This dropdown is highlighted with a red border.  
Below it is a field labeled 'Authentication URL (optional)' containing the URL 'https://services.think.ibm:1443/'.  
In the 'Authorization' section, there is a dropdown menu labeled 'Authorize application users using' with the value 'Authenticated' selected.  
At the bottom right of the interface are two buttons: 'Cancel' and 'Save'.

- \_\_\_ h. **Save** the changes.  
\_\_\_ i. Click **Confirm** in the dialog to update the API assembly.  
\_\_\_ j. Click the return arrow to return to the Resources page.

- 
- \_\_\_ 6. Update the OAuth provider in the Sandbox catalog.  
\_\_\_ a. Click the Manage option from the navigation menu in API Manager.  
\_\_\_ b. Click the **Sandbox** tile. Then, click **Settings**.

- \_\_ c. Click **OAuth Providers**. Then, click **Edit**.

Manage / Sandbox  
Settings

Overview  
Gateway Services  
Lifecycle Approvals  
Roles  
Onboarding  
API User Registries  
**OAuth Providers**

**OAuth Providers**  
Manage the OAuth Providers configured for API Manager

**Edit**

| TITLE                                                                              | TYPE           |
|------------------------------------------------------------------------------------|----------------|
|  | No items found |

- \_\_ d. Select oauth-provider.

| TITLE                                              | TYPE   |
|----------------------------------------------------|--------|
| <input checked="" type="checkbox"/> oauth-provider | Native |

**Global Knowledge®**

**Cancel** **Save**

Click **Save**.

- \_\_\_ 7. Update the authentication URL user registry for the Sandbox catalog.
- \_\_\_ a. Click **API User Registries** from the Sandbox settings page. Then, click **Edit**.

| TITLE | TYPE | SUMMARY        |
|-------|------|----------------|
|       |      | No items found |

- \_\_\_ b. Select the **SampleAuthURL** authentication URL.

| <input checked="" type="checkbox"/> TITLE         | TYPE               | SUMMARY                                                                                                                                    |
|---------------------------------------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <input checked="" type="checkbox"/> SampleAuthURL | Authentication URL | Created by OAuth Provider configuration as a sample. Make sure to update the OAuth Providers using this sample with a valid User Registry. |

- \_\_\_ c. Click **Save**.  
The Sandbox catalog can now use the authentication URL registry.
- \_\_\_ d. Click the return arrow to return to the Manage page of the API Manager.

## 8.2. Configure OAuth 2.0 authorization in the inventory API

In the previous section, you defined an OAuth 2.0 Provider API. The OAuth Provider defines two services that restrict access to API operations in the `inventory` scope. The authorization service checks the user name and password encoded in the HTTP Basic authentication header.

In this section, you specify a **security definition** in the `inventory` API definition. The OAuth 2.0 security definition specifies the grant type and scope that the OAuth Provider expects.

- 1. Switch to the `inventory` API definition.
  - a. Click the Develop option in the navigation menu.
  - b. Click the `inventory` API definition in the list of APIs.

The screenshot shows a list of APIs under the heading 'APIs and Products'. There are three entries: 'financing-1.0.0', 'inventory-1.0.0', and 'logistics-1.0.0'. The 'inventory-1.0.0' entry is highlighted with a red box around its title and icon.

- 2. Create an OAuth security definition.
  - a. Click **Security Definitions**. Then, click **Add**.

The screenshot shows the 'inventory' API development interface. The 'Security Definitions' tab is selected. A table lists one security definition:

|                | NAME   | TYPE   | LOCATED_IN | ⋮ |
|----------------|--------|--------|------------|---|
| clientIdHeader | apiKey | header |            | ⋮ |

- b. Select the **OAuth2** radio button.

- \_\_\_ c. Type the OAuth security definition details to match the OAuth Provider.
- Name: oauth
  - Description: OAuth authorization settings for the inventory API
  - Type: OAuth2
  - OAuth Provider: oauth-provider
  - Flow: Resource owner password
  - Token URL: `https://$(api.endpoint.address)/oauth20/oauth2/token`
  - Scopes: inventory

|                               |                                                                                                      |
|-------------------------------|------------------------------------------------------------------------------------------------------|
| <b>Name</b>                   | oauth                                                                                                |
| <b>Description (optional)</b> | OAuth authorization settings for the inventory API                                                   |
| <b>Type</b>                   | <input type="radio"/> API Key <input type="radio"/> Basic <input checked="" type="radio"/> OAuth2    |
| <b>OAuth Provider</b>         | <br>oauth-provider |
| <b>Flow</b>                   | Resource owner password                                                                              |
| <b>Token URL</b>              | <code>https://\$(api.endpoint.address)/oauth20/oauth2/token</code>                                   |

- \_\_\_ d. Click **Save**.  
 The oauth security definition is added to the list.
- \_\_\_ 3. Apply the security definition to the operations in the inventory API.
- \_\_\_ a. Click the **Security** section of the inventory API definition. Then, click **Add**.

- \_\_ b. Select the `oauth`, `clientIdHeader`, and `inventory` security definitions to apply it to all API operations.

The screenshot shows a "SECURITY DEFINITIONS" dialog box. It contains three checked checkboxes: "oauth", "clientIdHeader", and "inventory". A blue "Save" button is located at the bottom right of the dialog.

Click **Save**.



### Information

The **oauth** security definition specifies how client applications invoke operations in the inventory API definition. Specifically, a client application must fulfill the requirements that are set in the `oauth` security definition to call an inventory API operation.

In this example, the client application must send an API operation request with the `inventory` scope. The inventory API expects a password OAuth 2.0 grant type. To access the API, clients retrieve an access token from the token service from the OAuth Provider API, at <https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token>.

---

Global Knowledge ®

## 8.3. Create a test application

In this section, you create the client ID and client secret for an application to test the OAuth function.

- \_\_\_ 1. Create a test application in the Sandbox catalog.
  - \_\_\_ a. Click the Manage option in the navigation menu.
  - \_\_\_ b. Click the **Sandbox** tile.
  - \_\_\_ c. Click **Applications** in the navigation menu.
  - \_\_\_ d. Click the ellipsis alongside the sandbox-test-app.

| TITLE                              | APPLICATION TYPE | CONSUMER ORGANIZATION | STATE   |
|------------------------------------|------------------|-----------------------|---------|
| > <a href="#">sandbox-test-app</a> | Production       | sandbox-test-org      | Enabled |

Credentials
  
[Create subscription](#)

Click **Credentials**.

- \_\_\_ e. The client ID is added.

| TITLE                           | CLIENT ID                        |
|---------------------------------|----------------------------------|
| Sandbox Test App<br>Credentials | 27aa227243f4aaead72e18757f77de48 |

- \_\_\_ f. Click **Add**.

- g. The Client ID and Client Secret values are added.

## Credentials

Save the client secret (it will no longer be retrievable for security purposes)

Title

262f0648-dd0b-4b99-8886-ef5ed135ebbf

Client ID

5683a9b8ecb46763dcf6f168cfaee123

**Copy**

Client Secret

dfa35b39155ec7db4b984aba4f46c7c7

**Copy**

**Cancel**

**Create**

- h. Save the Client ID and Client secret values to be used later.



### Important

You **must** copy these values as they are retrievable later for security purposes.

- i. Open a terminal and type:  
`gedit inventory-cred`

Copy the client ID and client secret into the editor. Then, **save** the file.

You copy these values from the `inventory-cred` file later.

- j. Click **Create**.

- k. The credential is added.

| Credentials                          |                                  | <b>Add</b> |
|--------------------------------------|----------------------------------|------------|
| TITLE                                | CLIENT ID                        |            |
| 6ffa338f-af33-472a-89da-47fef3bc3abf | f061d5701fef6a75fe210a3dfc9b7d0f | ⋮          |
| Sandbox Test App Credentials         | 27aa227243f4aaead72e18757f77de48 | ⋮          |

- l. Click the return arrow twice to return to the Manage page.

## 8.4. Test OAuth security in the inventory API

In this section, you test the OAuth security that is added to the inventory API.

- \_\_ 1. Open a terminal window and start the LoopBack “back-end” application.

- \_\_ a. Navigate to the inventory application directory.

```
$ cd ~/inventory
$ pwd
/home/localuser/inventory
```

- \_\_ b. Start the Loopback application.

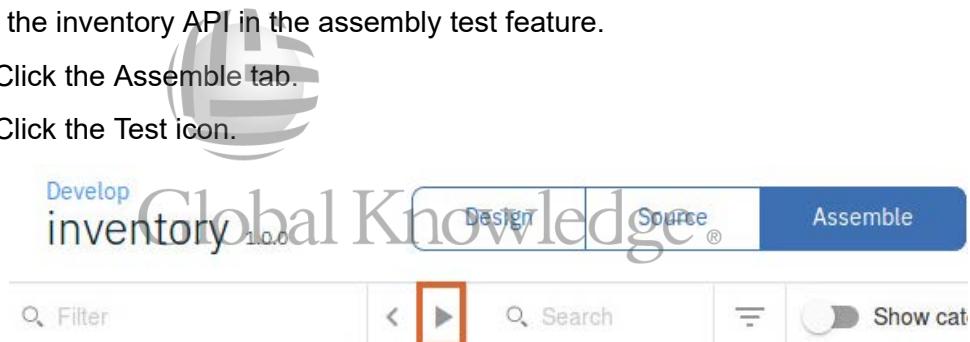
```
$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- \_\_ 2. Open the inventory API in the editor.

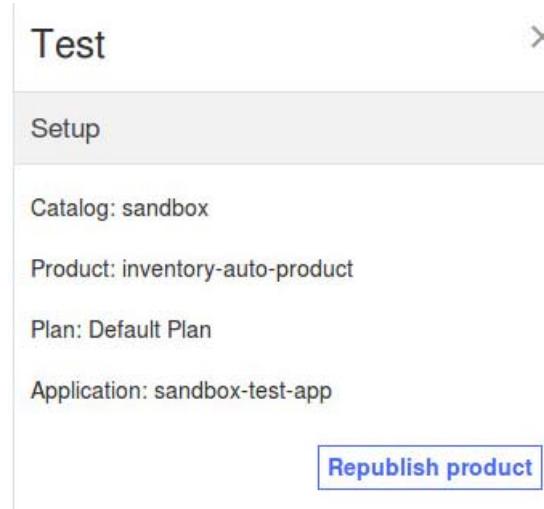
- \_\_ a. Click the Develop option in the navigation menu.
  - \_\_ b. Click the **Inventory** API in the list of APIs and Products.

- \_\_ 3. Test the inventory API in the assembly test feature.

- \_\_ a. Click the Assemble tab.
  - \_\_ b. Click the Test icon.



- \_\_\_ c. Click **Republish product**.



The Product is republished.

- \_\_\_ d. Select `get /items` from the operation drop-down list.  
 \_\_\_ e. The client ID and client secret are automatically inserted for you.  
 \_\_\_ f. Type the user credentials:

- Username: `user`
- Password: `pass`

This operation is secured with password flow OAuth

Authorization

Username: `user`

Password: `***`

- \_\_\_ g. The scope is prefilled.

```
explorer_scopes
 inventory

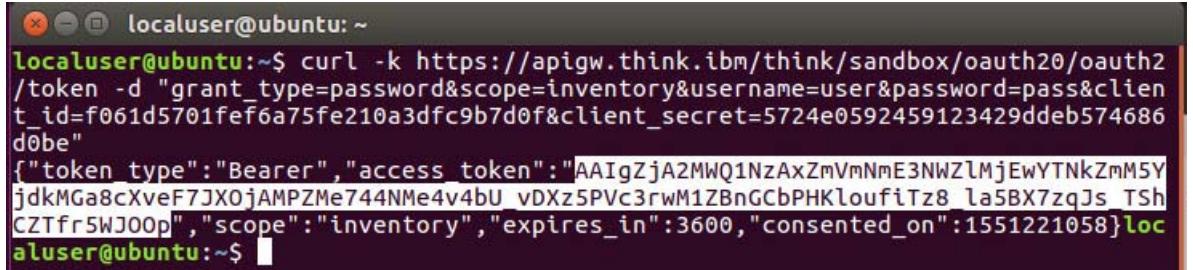
api_security_oauth_token_url:
https://$(api.endpoint.address)/oauth20/oauth2
/token
```

**explorer\_authorize**    **explorer\_access\_token**

\_\_ h. You must provide the explorer access token.

\_\_ i. Open a terminal window. Then, type:

```
curl -k https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token -d
"grant_type=password&scope=inventory&username=user&password=pass&client_id=<app_client_id>&client_secret=<app_client_secret>"
```



```
localuser@ubuntu:~$ curl -k https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token -d
"grant_type=password&scope=inventory&username=user&password=pass&client_id=f061d5701fef6a75fe210a3dfc9b7d0f&client_secret=5724e0592459123429ddeb574686
d0be"
{"token_type": "Bearer", "access_token": "AAIgZjA2MWQ1NzAxZmVmNmE3NWZLMjEwYTNkZmM5Y
jdkMGa8cXveF7JX0jAMPZMe744NMe4v4bU_vDXz5PVc3rwM1ZBnGCbPHKloufiTz8_la5BX7zqJs_TSh
CZTfr5WJ0Op", "scope": "inventory", "expires_in": 3600, "consented_on": 1551221058}loc
aluser@ubuntu:~$
```

You can use the template in `~/lab_files/authorization/oauth.txt` to create the curl command.

Replace the `<app_client_id>` and `<app_client_secret>` variables in the curl command template. Then, paste the command into the terminal.

\_\_ j. Copy the access token from the terminal window (everything between the quotation marks). Then, paste the token into the test feature in the explorer access token area.



\_\_ k. Scroll down in the test feature. Then, click **Invoke**.

- I. The response message is displayed in the test feature.

Invoke

Response

Status code:  
200 OK

Response time:  
415ms

Headers:

```
apim-debug-trans-id: 426530149-Landlord-
apiconnect-cea68102-0846-43b9-
afe7-65556c192757
content-type: application/json; charset=utf-8
```

Body:

```
[
 {
 "name": "Dayton Meat Chopper",
 "description": "Punched-card tabulating machines and time clocks were not the only products offered by the young IBM. Seen here in 1930, manufac
 }
]
```

- m. You successfully tested the inventory application with oauth security.
4. Close the test in the browser.
5. Stop the inventory application that is running in the terminal.
6. Sign out of API Manager and close the browser.



## Information

The Authentication URL in the exercise is a multi-protocol gateway policy that runs on the Services domain on the DataPower gateway. The policy accepts the URL ending with /authorize with any user ID and password and returns a 200 OK response message. Students with a knowledge of DataPower can sign on to the DataPower graphical interface and set the log option to debug and the probe option on for the Services MPGW. These students can then see the request and response messages in DataPower when they call the inventory application that is configured with OAuth security.

1. Sign on to the DataPower gateway.
  - a. In a browser, type `https://apigw.think.ibm:9090/`
  - b. Then, type:
    - User name: `admin`
    - Password: `passw0rd`
    - Domain: `Services`

Click **Login**.

Do not save any changes to the apiconnect domain. API Connect manages the apiconnect domain.

---

## End of exercise



## Exercise review and wrap-up

The first part of the exercise explained how to create and publish an OAuth 2.0 Provider: a set of API operations that authorize access to protected resources. You create the OAuth Provider that defined two operations: the `authorize` and `token` services.

The second part of the exercise covered how to secure API resources with an OAuth 2.0 message flow. You defined an OAuth 2.0 security requirement on all operations in the inventory API.





Global Knowledge®

# Exercise 9. Deploy an API implementation to a container runtime environment

## Estimated time

00:30

## Overview

In this exercise, you deploy the inventory LoopBack application to a Docker container runtime environment. You test that the inventory application that runs in the Docker container.

## Objectives

After completing this exercise, you should be able to:

- Test a local copy of a LoopBack API application
- Examine the Dockerfile that is used to deploy a Loopback API
- Create a Docker image by using the docker build command with the Dockerfile
- Verify that the API runs on the Docker image.

## Introduction

You defined, developed, and secured the inventory API with the API Connect Toolkit. You tested the application that runs natively with npm on the student workstation. You tested the application with the LoopBack API Explorer. You also tested the inventory application by calling it from the API Gateway. You now want to make your application more portable than running it as a Node application on the local workstation. You achieve this benefit by containerizing the application.

In this exercise, you learn how to deploy the API application to a Docker containerized runtime. For this exercise, the Docker image runs on the Ubuntu host workstation. Ideally, you then deploy the Docker image to the Kubernetes runtime that runs the API Connect software and is also running on the student host. The Kubernetes environment for the course exercises consists of a single master image. In a real-world configuration, the Kubernetes runtime includes a master node and worker nodes that run on different hosts in your network. The Kubernetes runtime provides the flexibility and scalability features for running your LoopBack application. You do not deploy the Docker image to the Kubernetes environment in this course.

**N**

You can deploy the LoopBack API application to any server environment that hosts Node.js and npm applications, such as a Node.js Cloud Foundry application buildpack or a Docker container. As you see in the exercise, the LoopBack application is deployed to a Docker image that runs a lightweight Alpine Linux base image with Node.js, npm, and the inventory application.

Deploying a Loopback application to a container runtime might not be the role of an API developer, but the task of an administrator or the owner of the provider organization. This information is provided to make you aware of the options for deploying a LoopBack application in API Connect.

---

## Requirements

Before you start this exercise, you must complete the data sources, remote, policies, and authorization exercises in this course.

You must complete this lab exercise in the student development workstation: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



## Exercise instructions

### Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



Global Knowledge®

## 9.1. Test a local copy of the LoopBack API application

In a previous exercise, you created the `inventory` LoopBack API application. You designed and implemented two model objects: `items` and `reviews`. The LoopBack framework created API paths and operations that map to create, retrieve, update, and delete functions on `items` and `reviews`.

Before you deploy and publish the `inventory` LoopBack API application, make sure that the application runs correctly on your workstation. You tested the application in an earlier exercise, so there should not be any issues with running the application.

- 1. Start the `inventory` application.
  - a. Open a terminal window.
  - b. Change directory to the `inventory` directory.  
`$ cd ~/inventory`  
`$ pwd`  
`/home/localuser/inventory`
  - c. Start `inventory` LoopBack application.  
`$ npm start`  
`> inventory@1.0.0 start /home/localuser/inventory`  
`> node .`  
`Web server listening at: http://0.0.0.0:3000`  
`Browse your REST API at http://localhost:3000/explorer`
  - d. Wait until the web server is started and listening.



I

By default, the LoopBack application listens on port 3000 when you start the application locally on your workstation. You can override this setting in the `server/config.json` configuration file.

- 
- 2. Test the GET `/inventory/items` API operation with the cURL utility.
    - a. In the Terminal application, click **File > Open Terminal**.

- \_\_\_ b. Make an HTTP GET request to the `/inventory/items` path.

```
$ curl -k -I http://0.0.0.0:3000/inventory/items
HTTP/1.1 200 OK
Vary: Origin, Accept-Encoding
Access-Control-Allow-Credentials: true
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
X-Download-Options: noopen
X-Content-Type-Options: nosniff
Content-Type: application/json; charset=utf-8
Content-Length: 10476
ETag: W/"28ec-0eqf3uQDRW98EMjba17CbQ"
Date: Wed, 20 Feb 2019 18:49:17 GMT
Connection: keep-alive
```



I

The **cURL** application is a highly customizable command line utility that makes HTTP and HTTPS requests to web servers. In this step, you called the `/inventory/items` API path on the local copy of the **inventory** LoopBack application.

The `-k` option instructs cURL to ignore security warnings from TLS certificate errors. By default, the cURL application does not accept self-signed security certificates. You ignore this warning in development and testing.

The `-I` or `--head` option instructs cURL to display the HTTP response message header, but not the message body. You can omit this parameter to see the list of inventory items from the service.



A

If you did not receive an HTTP status code of 200 OK, the inventory LoopBack API is not working correctly. Make sure that you correct any errors before you continue with this exercise.

- \_\_\_ 3. Stop the inventory application.
- \_\_\_ a. Switch to the Terminal window with the LoopBack API application.
  - \_\_\_ b. Press Ctrl+C to stop the Node application.
  - \_\_\_ c. You can close one of the Terminal windows.

## 9.2. Set up the Docker image configuration

With Docker technology, you can publish Loopback applications to a containerized environment.

- A Docker container image is an executable package that includes everything that is needed to run it: code, runtime, system libraries, and settings.

If you decide to use a Docker container to host your LoopBack application, you must use Docker software to build the Docker image. For the purposes of this exercise, Docker and docker-compose software are preinstalled on the workstation.

1. Build the Docker image.

- a. Open a Terminal application window.
- b. Ensure that you are in the inventory directory.  
\$ cd ~/inventory
- c. Copy the Dockerfile from the ~/lab\_files/containers folder.  
\$ cp ~/lab\_files/containers/Dockerfile ./
- d. Review the Dockerfile in the inventory folder.  
\$ cat Dockerfile

```
Build with:
docker build -t apic-inventory-image .
Create a small alpine linux distribution as the base image
FROM alpine:3.8
RUN apk add --update nodejs nodejs-npm && npm install npm@5.6.0 -g
RUN which node; node -v
WORKDIR /inventory
Copy the files in the host current dir to the Docker WORKDIR
ADD . /inventory
RUN pwd;ls
Make port available outside the container
EXPOSE 3000
#CMD to start
CMD ["npm", "start"]
Run the image after the build with the command:
docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image
```

- \_\_\_ e. Build the Docker image by using the Dockerfile.

```
$ docker build -t apic-inventory-image .
```

```
Sending build context to Docker daemon 97.5 MB
Step 1/8 : FROM alpine:3.8
--> 76da55c8019d
Step 2/8 : RUN apk add --update nodejs nodejs-npm && npm install npm@5.6.0 -g
--> Running in 129df0759f46
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/community/x86_64/APKINDEX.tar.gz
(1/9) Installing ca-certificates (20171114-r3)
...
Step 7/8 : EXPOSE 3000
--> Running in 9cadad965d66
--> 09de2c5ce14f
Removing intermediate container 9cadad965d66
Step 8/8 : CMD npm start
--> Running in 77c82583fcc8
--> e743a15b9563
Removing intermediate container 77c82583fcc8
Successfully built e743a15b9563
```



The Docker build command includes a period (.) at the end for the path that the build command uses. The path option specifies that all the files in the local directory are sent to the Docker daemon when the build command is run.

If you are unable to build the Docker image, refer to "[Issues when building the Docker image](#)" on page B-3 in [Appendix B, "General troubleshooting issues when working on the exercises"](#).

- \_\_\_ f. Display the list of Docker images.

| REPOSITORY           | TAG         | IMAGE ID     | CREATED       |
|----------------------|-------------|--------------|---------------|
| SIZE                 |             |              |               |
| apic-inventory-image | latest      | e743a15b9563 | 7 minutes ago |
| 136 MB               |             |              |               |
| alpine               |             | 3.8          |               |
| 491e0ff7a8d5         | 3 weeks ago | 4.41MB       |               |

- \_\_\_ g. Review the content of the image.

```
$ docker run -it apic-inventory-image /bin/sh
/inventory # ls
Dockerfile common node_modules package.json
client inventory.yaml package-lock.json server
/inventory # node -v
v8.14.0
/inventory # exit
$ docker ps -a
$ docker rm <container ID>
```



I

You can use the first few characters of the container ID as a shorthand notation when you remove the Docker container. Make sure that you remove only the container that runs the inventory image. The other K8 containers are part of the API Connect runtime environment.

The contents of the inventory directory is copied to the apic-inventory-image Docker image. In the lab environment, the apic-inventory-image listens to commands on port 3000.

- \_\_\_ 2. Run the application on the Docker image. You do not need to run the command from the ~/inventory directory.

```
$ cd ~
```

- \_\_\_ a. In the terminal, type:

```
$ docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image
```

 Global Knowledge®

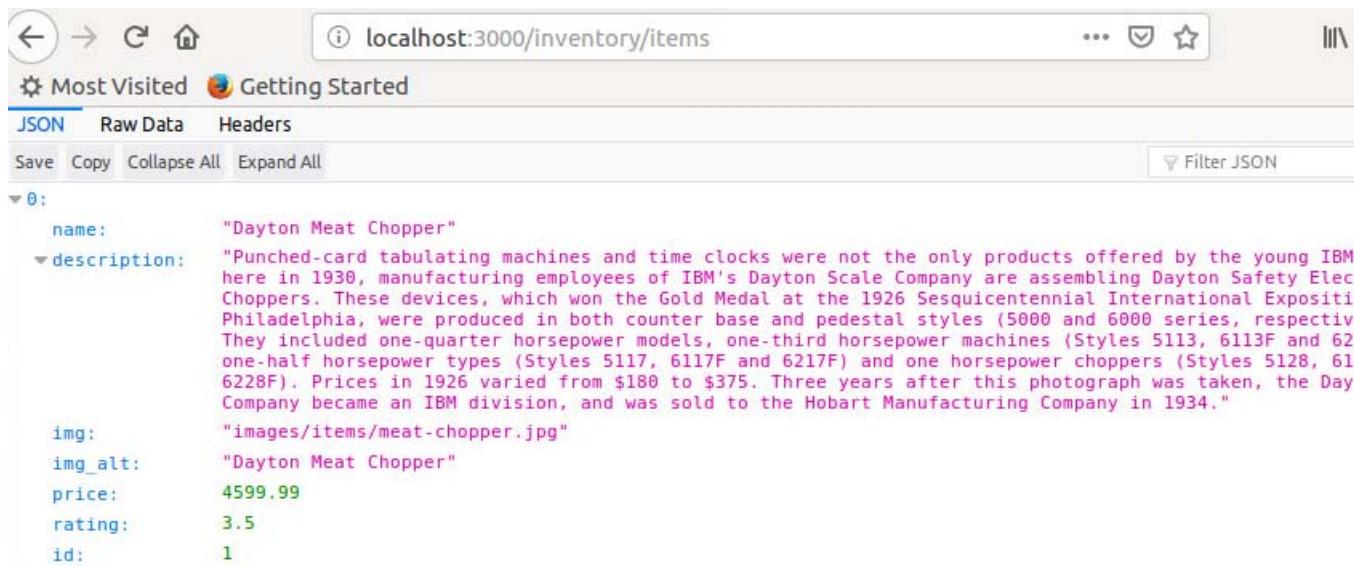
```
> inventory@1.0.0 start /inventory
> node .
```

Web server listening at: http://localhost:3000  
Browse your REST API at http://localhost:3000/explorer

- \_\_\_ 3. Review the contents of the /inventory/items API operation.

- \_\_\_ a. Open a web browser.  
\_\_\_ b. Enter <http://localhost:3000/inventory/items> in the address bar.

- \_\_\_ c. Confirm that a list of inventory items appears in the web page, starting with the Dayton Meat Chopper.



A screenshot of a browser window displaying JSON data for an inventory item. The URL in the address bar is `localhost:3000/inventory/items`. The JSON object is as follows:

```

{
 "name": "Dayton Meat Chopper",
 "description": "Punched-card tabulating machines and time clocks were not the only products offered by the young IBM here in 1930, manufacturing employees of IBM's Dayton Scale Company are assembling Dayton Safety Electric Choppers. These devices, which won the Gold Medal at the 1926 Sesquicentennial International Exposition in Philadelphia, were produced in both counter base and pedestal styles (5000 and 6000 series, respectively). They included one-quarter horsepower models, one-third horsepower machines (Styles 5113, 6113F and 6217F) and one-half horsepower types (Styles 5117, 6117F and 6217F) and one horsepower choppers (Styles 5128, 61228F). Prices in 1926 varied from $180 to $375. Three years after this photograph was taken, the Dayton Safety Electric Company became an IBM division, and was sold to the Hobart Manufacturing Company in 1934.",
 "img": "images/items/meat-chopper.jpg",
 "img_alt": "Dayton Meat Chopper",
 "price": 4599.99,
 "rating": 3.5,
 "id": 1
}

```

- \_\_\_ 4. Stop the running the Docker container.

- \_\_\_ a. Open another terminal window or tab.  
 \_\_\_ b. In the terminal, type:

```
$ docker ps
```

| CONTAINER ID  | IMAGE                  | COMMAND             | CREATED        |
|---------------|------------------------|---------------------|----------------|
| STATUS        | PORTS                  | NAMES               |                |
| 65bce9862bd9  | apic-inventory-image   | "npm start"         | 17 minutes ago |
| Up 17 minutes | 0.0.0.0:3000->3000/tcp | compassionate_knuth |                |

- \_\_\_ c. In the terminal, type:

```
$ docker stop <container ID>
```



N

You can use the first few characters of the container ID as a shorthand notation when you stop the Docker container. The docker run command also completes in the other terminal window.

- \_\_\_ 5. Close the terminal windows.

## End of exercise

## Exercise review and wrap-up

The first part of the exercise reviews the inventory API application that you built in the previous exercise. You confirmed that the LoopBack application works in your workstation environment.

In the second part of the exercise, you set up the Loopback application on a Docker image by using a Dockerfile.

You tested the LoopBack application that runs on a Docker image.



# Exercise 10. Define and publish an API product

## Estimated time

00:30

## Overview

This exercise examines how to publish APIs with plans and products. You create a product and a plan, and deploy the product in API Manager.

## Objectives

After completing this exercise, you should be able to:

- Create a product and plan in the API Manager
- Add the APIs to the product
- Verify that the invoke URL for the inventory application routes to the Docker image
- Publish the product to the Sandbox catalog

## Introduction

In a previous exercise, you published the implementation of an API: the `inventory` LoopBack application. In this exercise, you package and publish the `financing`, `logistics`, and `inventory` API definitions into a single product in API Manager.

- An **API definition** lists the paths and operations in an API. For each operation, the API definition specifies the possible request, response, and fault message. API definitions also contain metadata about an API, including version, description, security configuration, environment properties, and message processing policies.
  - An OpenAPI definition file is the design-time artifact that represents the API definition.
- A **product** combines one or more API definitions into a bundle. The product itself includes metadata, version, and licensing information.
  - A product document is the design-time artifact that represents an API product. Unlike the OpenAPI definition file, this document is not part of a standard. The source for the product document is defined in YAML format.
- A **plan** is a contract between the API provider and the API consumer. It specifies the rate of API calls over a defined time period. A plan is defined in a section of a product.

When you publish a product, you make the API definitions in the product available for use. As part of the publish process, the API gateway configures network endpoints according to the API

definition. The gateway enforces security constraints and processes messages according to policies in the API definition.

The product, API definition, and plans also appear in the Developer Portal. Application developers who want to use or consume APIs can retrieve the API definition from the portal.

## Requirements

Before you start this exercise, you must complete the data sources, remote, policies, authorization, and container exercises in this course.

You must complete this lab exercise on the student image: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



## Exercise instructions

### Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.

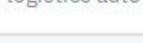


Global Knowledge®

## 10.1. Create a product for all three APIs

In this section, you explicitly create a product that packages the financing, inventory, and logistics APIs into a bundled offering.

- \_\_\_ 6. If you are not already signed on, open the API Manager web application.
  - \_\_\_ a. Open a browser window.  
Then, type `https://manager.think.ibm.`
  - \_\_\_ b. Sign in to API Manager. Type the credentials:
    - User name: ThinkOwner
    - Password: Passw0rd!
 Click **Sign in**.  
You are signed in to API Manager.
- \_\_\_ 7. Review the state of the existing APIs and products in API Manager.
  - \_\_\_ a. Click the Develop APIs and Products tile from the API Manager home page, or select the Develop option from the navigation menu.
  - \_\_\_ b. The list of APIs and products is displayed.

| APIs and Products                                                                                                |            | Add ▾ |
|------------------------------------------------------------------------------------------------------------------|------------|-------|
|                                                                                                                  |            |       |
| TITLE                                                                                                            | TYPE       |       |
|  financing-1.0.0               | API (REST) |       |
|  inventory-1.0.0              | API (REST) |       |
|  logistics-1.0.0              | API (REST) |       |
|  financing auto product-1.0.0 | Product    |       |
|  inventory auto product-1.0.0 | Product    |       |
|  logistics auto product-1.0.0 | Product    |       |



## Information

The inventory API was created by importing an OpenAPI definition into API Manager with the target URL as the LoopBack application.

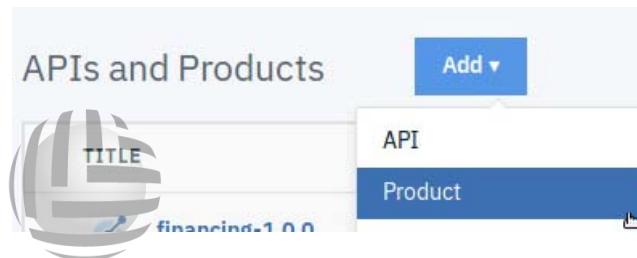
The financing API was created as a new OpenAPI definition in API Manager with the target URL being a SOAP service that runs in the DataPower Services domain.

The logistics API was also created as a new OpenAPI definition in API Manager where the target URL for the stores points to an external service that returns the location of a US postal code.

The financing auto product, financing auto product, and logistics auto product are product definitions that were auto-generated when the APIs were published in the test feature of the assembly in API Manager.

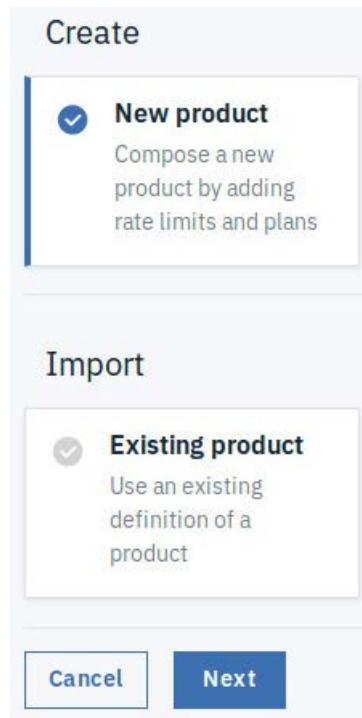
8. Create a single product for the APIs.

- a. On the Develop page, click **Add**.  
Then, select **Product**.



Global Knowledge ®

- \_\_ b. On the Add Product page, select **New Product**.



Click **Next**.



Global Knowledge ®

\_\_ c. Type the product information.

- Title: **think-product**
- Name: **think-product**
- Version **1.0.0**

**Info**

Enter details of the product

---

**Title**

think-product

**Name**

think-product

**Version**

1.0.0

**Summary (optional)**



Click **Next**.

\_\_ d. Select the check box alongside title to add the financing, inventory, and logistics APIs to the product.

**APIs**

Select the APIs to add to this product

| <input checked="" type="checkbox"/> | TITLE     | VERSION | DESCRIPTION |
|-------------------------------------|-----------|---------|-------------|
| <input checked="" type="checkbox"/> | financing | 1.0.0   |             |
| <input checked="" type="checkbox"/> | inventory | 1.0.0   |             |
| <input checked="" type="checkbox"/> | logistics | 1.0.0   |             |

Back
Cancel
Next

Click **Next**.

- \_\_ e. Accept the defaults to add a Default Plan to the product.

Plans

Add plans to this product

Add

Default Plan

Title

Default Plan

Description (optional)

Default Plan

Rate Limit

100 / 1 hour

Back Cancel Next

Click **Next**.



Global Knowledge ®

- \_\_ f. Accept the defaults for the publish, visibility, and subscribability options.

**Publish**

Enable publishing of this product

---

Publish product

---

**Visibility**

Select the organizations or groups you would like to make this product visible to

---

**Public**  
 **Authenticated**  
 **Custom**

---

**Subscribability**

Select the organizations or groups you would like to subscribe to this product

---

**Authenticated**  
 **Custom**

---

**Global Knowledge ®**

Click **Next**.

- \_\_ g. The think-product is added and the summary page is displayed.

The screenshot shows a 'Create New Product' interface with a 'Summary' section. It lists three completed tasks with green checkmarks:

- Created new product
- Added APIs
- Added rate limits

- \_\_ h. Click the return arrow to return to the Develop page.  
\_\_ i. The think-product is added to the list of APIs and products.

The screenshot shows a table of APIs and Products. The columns are TITLE, TYPE, and LAST MODIFIED. The table contains the following data:

| TITLE                        | TYPE       | LAST MODIFIED |
|------------------------------|------------|---------------|
| financing-1.0.0              | API (REST) | 8 days ago    |
| inventory-1.0.0              | API (REST) | a day ago     |
| logistics-1.0.0              | API (REST) | 8 days ago    |
| financing auto product-1.0.0 | Product    | 8 days ago    |
| inventory auto product-1.0.0 | Product    | a day ago     |
| logistics auto product-1.0.0 | Product    | 8 days ago    |
| think-product-1.0.0          | Product    | 3 minutes ago |



## Information

The **visibility** setting determines whether an application developer can see an API product on the Developer Portal. The **subscribable** setting determines which type of application developer can subscribe to the product.



Global Knowledge®

## 10.2. Verify that the target URL in the inventory API routes to the Docker image

In this section, you verify that the value of the invoke URL for the inventory API routes to the Docker image and port number. The application server is defined as a property in the API definition.

- 1. Open the inventory API definition in the editor.

  - a. Click the inventory API in the list of APIs and Products in the Develop page of the API Manager.

- 2. Review the API property named app-server.

  - a. Click **Properties** in the Design view.
  - b. The list of API properties is displayed.

| PROPERTY NAME | ENCODED | DESCRIPTION                               |
|---------------|---------|-------------------------------------------|
| target-url    | false   | The URL of the target service             |
| app-server    | false   | Host where the inventory application runs |

- c. Click **app-server** in the property list.
- d. The property opens in the editor.



### Information

Notice that the default value for the app-server property is  
<http://platform.think.ibm:3000>

The app-server property points to the host and port number where the inventory application runs. The inventory application can run either as a NodeJS application from the /home/localuser/inventory directory or as a docker container with the command:

```
$ docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image
```

The docker run command maps the container port 3000 to the local port 3000.

The app-server default value facilitates calling either the local NodeJS application or the Docker container. No changes are required.

---

- e. Click **Cancel** to exit the editor without changes.
- f. Select the Develop option from the navigation menu.



## 10.3. Enable API key security in the financing and logistics API

Earlier in this exercise, you set the API key check off in the financing and logistics APIs to test the operations. Before you publish the think-product, enable API key security in the APIs. When you call an operation in the financing and logistics API, you must provide a valid client ID and client secret value.

- \_\_\_ 1. In the Develop page, click the **financing** API definition in the list of APIs and Products to edit the API.
- \_\_\_ 2. Enable API key security in the financing API.
  - \_\_\_ a. In the financing API definition, select the **Security** section in the Design view.

The screenshot shows the IBM API Connect interface. At the top, there's a navigation bar with 'Develop' and 'financing 1.0.0'. Below it, tabs for 'Design', 'Source', and 'Assemble' are visible, with 'Design' being the active one. On the left, a sidebar has 'API Setup', 'Security Definitions', 'Paths', and 'Security' (which is highlighted in blue). The main content area is titled 'Security' and contains the message 'No security definitions are defined.' There's also a note: 'Security definitions selected here apply across the API, but can be overridden for individual operations. [Learn more](#)' and a blue 'Add' button.

- \_\_\_ b. Click **Add**.
- \_\_\_ c. Select the `clientIdHeader apiKey` security requirement.

The screenshot shows a modal dialog titled 'SECURITY DEFINITIONS'. It contains a single checkbox labeled 'clientIdHeader apiKey' which is checked. A blue 'Save' button is at the bottom right of the dialog.

Click **Save**.

The API is updated.

- \_\_\_ d. Click the Develop option in the navigation menu to return to the list of APIs and products.
- \_\_\_ 3. In the Develop page, click the **logistics** API definition in the list of APIs and Products to edit the API.

- \_\_\_ 4. Enable API key security in the logistics API.
  - \_\_\_ a. In the logistics API definition, select the **Security** section in the Design view.

The screenshot shows the API Management interface for a product named "logistics 1.0.0". The top navigation bar has tabs for "Develop", "Design" (which is selected), "Source", and "Assemble". Under the "Design" tab, there are sections for "API Setup", "Security Definitions", "Security" (which is selected and highlighted in blue), and "Paths". The "Security" section contains a sub-section titled "SECURITY DEFINITIONS". A message states: "Security definitions selected here apply across the API, but can be overridden for individual operations. [Learn more](#)". A blue "Add" button is visible on the right. Below this, a message says "No security definitions are defined."

- \_\_\_ b. Click **Add**.
    - \_\_\_ a. Select the `clientIdHeader apiKey` security requirement.
    - \_\_\_ b. Enable the `clientIdHeader apiKey` security requirement.
- 
- The screenshot shows a modal dialog box titled "SECURITY DEFINITIONS". It contains a list of requirements: "clientIDHeader" and "apiKey", with the "clientIDHeader" option checked (indicated by a checked checkbox icon). At the bottom right of the dialog is a blue "Save" button. The background of the main interface shows the "Global Knowledge" logo.
- \_\_\_ c. Click **Save**.  
The API is updated.
  - \_\_\_ d. Click the Develop option in the navigation menu to return to the list of APIs and products.

## 10.4. Publish the product and API definitions

To make the API products, plans, and definitions available for use, you must publish these resources in the API Manager.

- 1. Ensure that the Develop page is selected in API Manager.



### Information

When you publish an API product to the API Connect Cloud, you specify the `catalog` where you want to publish the product.

- The **catalog** represents a collection of products in an organization. You separate products and APIs for testing before you make them available to developers. This course has only one catalog that is named Sandbox. The Sandbox catalog is created by default when API Connect is installed. Sandbox is a development catalog and products with the same version number are overwritten when they are published.

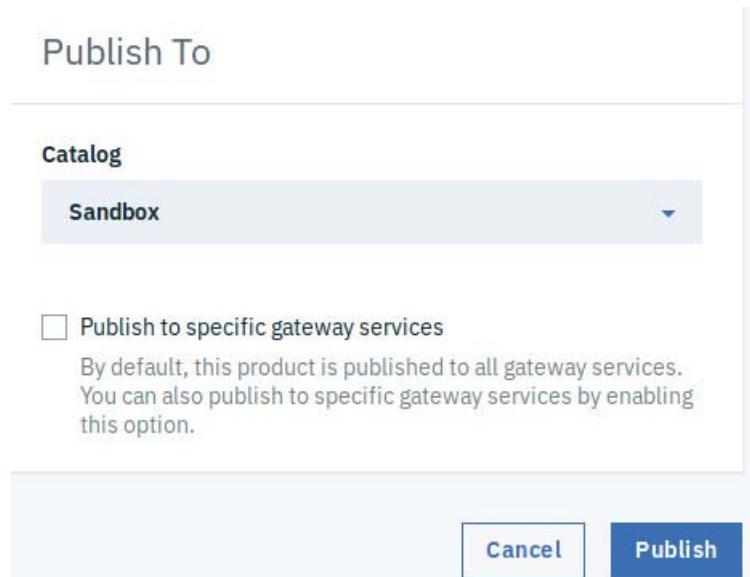
- 2. Publish the API product to the API Connect Cloud Sandbox catalog.

- a. In the list of APIs and Products, select the ellipsis option alongside the `think-product`. Then, select **Publish**.

| TITLE                        | TYPE       | LAST MODIFIED  |
|------------------------------|------------|----------------|
| financing-1.0.0              | API (REST) | an hour ago    |
| inventory-1.0.0              | API (REST) | a day ago      |
| logistics-1.0.0              | API (REST) | 32 minutes ago |
| financing auto product-1.0.0 | Product    | 8 days ago     |
| inventory auto product-1.0.0 | Product    | a day ago      |
| logistics auto product-1.0.0 | Product    | 8 days ago     |
| think-product-1.0.0          | Product    | 3 hours ago    |

Items per page: 50 | 1-7 of 7 items      1 of 1 pages      <      >      Stage      Publish

- \_\_\_ b. On the publish to page, select the Sandbox catalog. Then, click **Publish**.



- \_\_\_ c. The product is published to the Sandbox catalog.



### Information

What is the effect of the publish option?

When you select **publish**, you copy the API product and all of its associated API definitions to the selected catalog on the API Management server. The product and APIs are published to the Developer Portal and API Gateway.

In this exercise, you publish the think-product and all API definitions that belong to the product: the inventory, financing, and logistics API definitions. You do not publish the inventory application. The inventory application is managed externally.

- \_\_\_ 3. Confirm that the think-product is successfully published to API Manager.
- \_\_\_ a. Click the Manage option from the navigation menu.
- \_\_\_ b. Click the Sandbox catalog tile.  
The list of published products is displayed. Notice that the think-product is published.

- \_\_\_ c. Click the chevron alongside think-product.

Manage / Sandbox

## Products

| TITLE                    | NAME                         | STATE     |   |
|--------------------------|------------------------------|-----------|---|
| > financing auto product | financing-auto-product 1.0.0 | Published | ⋮ |
| > inventory auto product | inventory-auto-product 1.0.0 | Published | ⋮ |
| > logistics auto product | logistics-auto-product 1.0.0 | Published | ⋮ |
| <b>➤ think-product</b>   | think-product 1.0.0          | Published | ⋮ |

- \_\_\_ d. The details of the status of the product, APIs, and plans are displayed.

|                         |                     |                       |        |
|-------------------------|---------------------|-----------------------|--------|
| ▼ think-product         | think-product 1.0.0 | Published             | ⋮      |
| <b>APIs</b>             |                     |                       |        |
| financing:1.0.0         |                     |                       | online |
| inventory:1.0.0         |                     |                       | online |
| logistics:1.0.0         |                     |                       | online |
| <b>PLANS</b>            |                     |                       |        |
| Default Plan            |                     |                       |        |
| <b>GATEWAY SERVICES</b> |                     | <b>GATEWAY TYPE</b>   |        |
| DataPower API Gateway   |                     | DataPower API Gateway |        |

- \_\_\_ 4. Click the return arrow to return to the Manage page.

## End of exercise

## Exercise review and wrap-up

In the first part of the exercise, you created an API product: a collection of API definitions that are grouped for deployment. You also defined an API plan: a set of non-functional requirements that govern the rate and limit of API calls from the consumer. You added the API definitions to the product.

In the final part of the exercise, you published the API product, plan, and definitions to the Sandbox catalog on the API Management server.





Global Knowledge®

# Exercise 11. Subscribe and test APIs

## Estimated time

01:00

## Overview

In this exercise, you learn about the application developer experience in the Developer Portal. You review the consumer organization that is created for you. You sign on to the Developer Portal as the owner of the consumer organization. You review the published products and APIs. You register an application that uses the product and APIs. You review the client ID and client secret values, subscribe to an API plan, and test operations from an API product. Finally, you test all the APIs from a web-based consumer application.

## Objectives

After completing this exercise, you should be able to:

- Review the consumer organizations of the Sandbox catalog in API Manager
- Review the portal settings for the Sandbox catalog
- Sign on to the Sandbox catalog Developer Portal as the owner of the consumer organization
- Register an application in the Developer Portal
- Review the client ID and client secret values
- Test API operations in the Developer Portal
- Test all the APIs that you created with a web-based consumer application

## Introduction

In the previous exercises in this course, you assumed the role of the API developer: the provider of the API. The API developer defines, implements, secures, and tests the API application.

In this exercise, you focus on the application developer role: the person that creates mobile or web applications that use or consume the APIs. The application developer registers and develops an application that calls the operations in the published API.

A user account on the Developer Portal is already created for you: the owner of the `Publish` developer organization is already created in API Manager. You sign on to the Developer Portal with this user account. You register your client application in your account. The Developer Portal generates the client ID and client secret metadata that uniquely identifies your application when you make API calls. You test the `financing` and `logistics` API operations from the test client in the Developer Portal. In a final step, you test a JavaScript client application with the OAuth secured `/inventory/items` API operation.

## Requirements

Before you start this exercise, you must complete the data sources, remote, policies, authorization, containers, and publish exercises in this course.

You must complete this lab exercise on the student remote image: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



## Exercise instructions

### Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



Global Knowledge®

## 11.1. Review the consumer organizations in the Sandbox catalog

Consumer organizations include the users who sign up to use the Developer Portal. The owner of the think provider organization has the permission to manage consumers. A consumer organization is already created in the Sandbox catalog. You review the consumer organizations in this part.

- \_\_\_ 5. If you are not already signed on, open the API Manager web application.
  - \_\_\_ a. Open a browser window.  
Then, type `https://manager.think.ibm.com`.
  - \_\_\_ b. Sign in to API Manager. Type the credentials:
    - User name: ThinkOwner
    - Password: Passw0rd!
 Click **Sign in**.  
You are signed in to API Manager.
  
- \_\_\_ 6. Review the consumer organizations for the Sandbox catalog in API Manager.
  - \_\_\_ a. Click the Manage Catalogs tile from the API Manager home page, or select the Manage option from the navigation menu.
  - \_\_\_ b. Click the **Sandbox** tile to open the Sandbox settings.
  - \_\_\_ c. Click Consumer Organizations from the navigation menu.



- \_\_\_ d. The list of consumer organizations is displayed.

| TITLE                     | OWNER                                    | STATE   |
|---------------------------|------------------------------------------|---------|
| Publish                   | OP<br>Owner Publish<br>owner@publish.org | Enabled |
| Sandbox Test Organization | TU<br>Test User                          | Enabled |



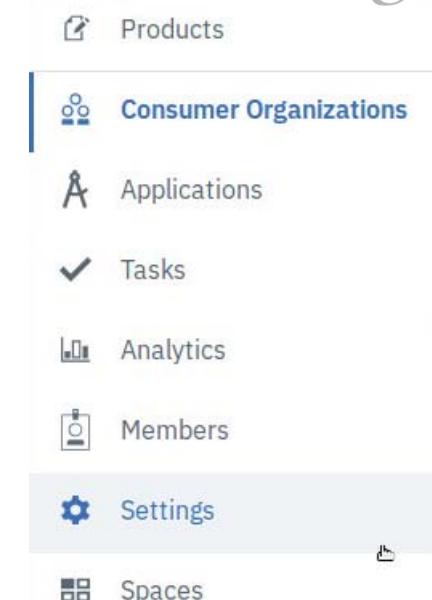
### Information

The Sandbox Test Organization is used when you test your APIs with the test feature of the API assembly. The Sandbox Test Organization is created when the API Connect product is installed.

The Publish consumer organization was created after product installation. You use the owner of the Publish organization to sign on to the Developer Portal.

- \_\_\_ 7. Review the portal settings for the Sandbox catalog

- \_\_\_ a. Click the Settings options in the navigation menu.



- \_\_\_ b. From the settings page, click **Portal**.

- \_\_\_ c. The portal settings are displayed.

The screenshot shows a configuration page for a developer portal. At the top, it says "Portal" and "Configure the developer portal that is used by application developers to access the APIs in this catalog". Below this, there are three sections: "Portal Service" (set to "Portal Service"), "Portal URL" (set to "https://portal.think.ibm/think/sandbox"), and "User Registries" (set to "Sandbox Catalog User Registry").

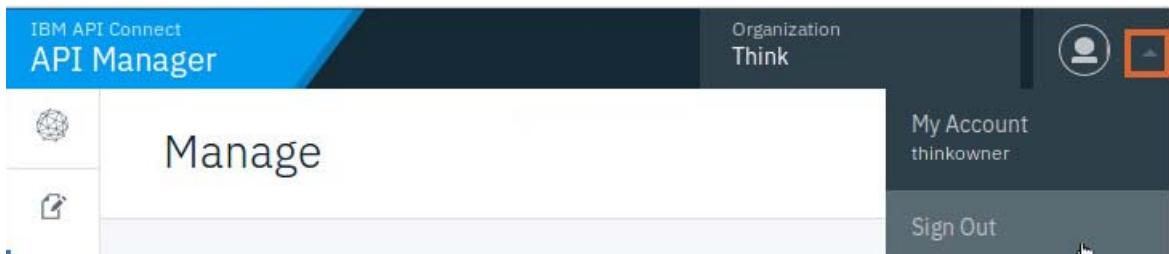
## Information

The portal settings show the configuration of the portal. The portal service uses the URL `https://portal.think.ibm/think/sandbox`. This address is used to sign on to the Developer Portal. Also, notice that the portal uses an API Connect local user registry that is named the Sandbox Catalog User Registry. All portal users are stored in this registry.

The Portal URL is a concatenation of these values:

- Portal host address
- Provider organization
- Catalog name

- \_\_\_ d. Click the return arrow to return to the Manage page.  
\_\_\_ 8. Sign out of the API Manager.



## 11.2. Register a client application in the Developer Portal

In this section, you sign on to the developer portal with a user ID that is already created for you.

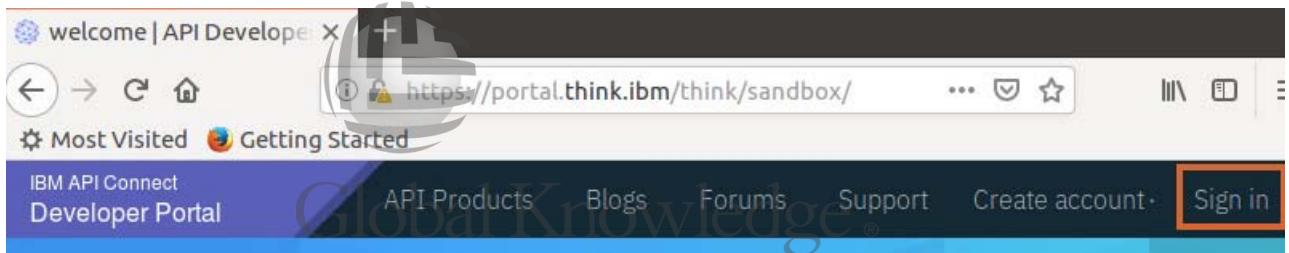
- \_\_\_ 1. Open the Developer Portal page.
  - \_\_\_ a. In a web browser address, type:  
`https://portal.think.ibm/think/sandbox`
  - \_\_\_ b. The Developer Portal public page is displayed.



### Note

Notice that some of the auto-published products are displayed on the public portal interface. These products are published with visibility of public.

- \_\_\_ 2. Sign in to the Developer Portal with the account of the owner of the **Publish** consumer organization.
  - \_\_\_ a. Click the **Sign-in** link.



- \_\_\_ b. Enter the user credentials for the account.

- User name: **OwnerPublish**
- Password: **Passw0rd!**

API Developer Portal

## Sign in

[Sign in with Sandbox Catalog User Registry](#)

Username

OwnerPublish

Password

.....

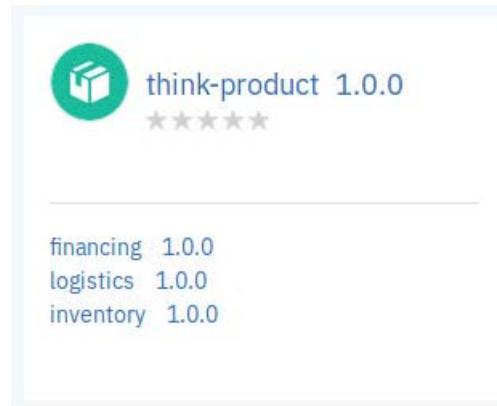
**Sign in**

Click **Sign in**.

- \_\_\_ c. The user is signed in to the Publish organization on the Developer Portal.  
\_\_\_ 3. Click the **See all products** link to display the list of published products.

The screenshot shows the IBM API Connect Developer Portal homepage. At the top, there's a purple header bar with the text "BM API Connect Developer Portal". To the right of this, there's a navigation bar with links for "API Products", "Apps", "Blogs", "Forums", and a search icon. Further to the right, it says "Organization Publish". Below the header, there's a large blue banner with a white abstract logo on the left. The text on the banner reads "brace yourselves. APIs are coming." and "Explore, subscribe to and be creative with our APIs. We can't wait to see what you come up with!". Below this banner, there's a purple button labeled "Explore API Documentation". At the bottom of the page, there are two buttons: "API Products" on the left and "See all products" on the right, which is highlighted with a red border.

- \_\_\_ 4. The think-product that you published earlier is displayed in the list of published products.



### Note

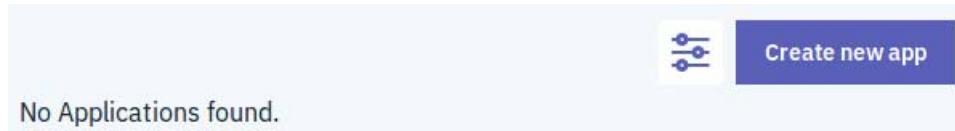
If you do not see the think-product in the list of API products, ensure that you navigate through all pages of products on the Developer Portal.



If you still do not see the product, you must publish the product. Refer to [Exercise 10, "Define and publish an API product,"](#) on page 10-1. Republish the product if necessary.

- \_\_\_ 5. Register a client application with the **OwnerPublish** account.

- \_\_\_ a. Click **Apps** from the navigation bar.
- \_\_\_ b. The account has no registered client applications.



Click **Create new App** to register one.

- \_\_\_ c. In the Create a new application page, type:
  - Title: **Think Application**

- \_\_\_ d. Description: A web application to browse items for sale, and to submit reviews.

## Create a new application

Title \*

Think Application

Description

A web application to browse items for sale, and to submit reviews.

Application OAuth Redirect URL(s)

|                                  |                        |
|----------------------------------|------------------------|
|                                  | X                      |
| <a href="#">Add another item</a> |                        |
|                                  | <a href="#">Cancel</a> |
|                                  | <a href="#">Submit</a> |

Click **Submit**.



The application is created.



Information

## Global Knowledge ®

The Developer Portal assigns a unique identifier, an API Key (client ID), to the client application that you registered. When you call APIs that are hosted on the API Connect Cloud, you must send the API Key in the HTTP request message header.

The Portal also assigns a secret passphrase, a client secret, for the client ID. As an extra layer of security, you send the client ID and client secret in the message header for API calls.

Like a password, you must take precautions to protect the client secret value. You should send a client secret value over a secured TLS/SSL connection.

- \_\_\_ 6. Save the API Key and Secret values in a file on the student image.
- \_\_\_ a. In the apps page, select the **Show** check boxes to display the key and secret values.

Application created successfully.

## API Key and Secret

The API Key and Secret have been generated for your application.

**Key**

010069709451404b2508e3b5cf187149  Show

**Secret**

22c490ee5b007f8ad19a8fbbe581e9b7  Show

The Secret will only be displayed here one time. Please copy your API Secret and keep it for your records.

**Continue**

- \_\_\_ b. Open the File Manager on the student image.
- \_\_\_ c. Go to the `inventory-cred` file that you saved in the home directory in an earlier exercise. Right-click the file. Then, select **Open with gedit**.
- 
- \_\_\_ d. Copy the values for both the API Key and secret from the application in the browser to the editor for the file.
- 
- \_\_\_ e. Save the changes to the file. Then, minimize the file.
- \_\_\_ f. Click **Continue** back in the browser of the Developer Portal.
- \_\_\_ g. The Think Application is displayed on the page with the Dashboards tab selected. Since the application has never been called, no metrics are shown for the API statistics.

## 11.3. Subscribe the think application to a product and plan

The Think Application must subscribe to a product and plan to use an API. In this section, you subscribe the application to a product and a plan.

- 1. Create a subscription for the application.
  - a. Click the **Subscriptions** tab on the Think Application page.
  - b. Go to the bottom of the page. Then, click the [Why not browse the available APIs](#) link.

The screenshot shows the IBM Cloud Think Application dashboard. At the top, there's a navigation bar with 'Dashboard' and 'Subscriptions'. Below it, under 'Credentials', there are fields for 'Client ID' (with a redacted value) and 'Client Secret' (with a redacted value). A 'Verify' button is visible. To the right, there's a 'Show' checkbox. Under 'Subscriptions', it says 'No subscriptions found.' and has a link 'Why not browse the available APIs?' which is highlighted with a red box.

The list of products is displayed.

- c. Click the link in the **think-product** tile in the list of published products.
- d. The list of APIs and plans is displayed. Only one available plan exists.

- \_\_ e. Click **Subscribe** in the Default Plan area.

The screenshot shows the 'think-product' application page. At the top, there is a green circular icon with a white cube, followed by the text 'think-product' and '1.0.0' with a five-star rating. Below this, there are two sections: 'APIs' and 'Plans'. The 'APIs' section contains three items: 'financing 1.0.0' (with a pink gear icon), 'logistics 1.0.0' (with an orange gear icon), and 'inventory 1.0.0' (with an orange gear icon). The 'Plans' section contains one item: 'Default Plan' (with a grey gear icon). Below the plan name is a blue 'Subscribe' button and a link 'View details ▾'.

- \_\_ f. On the Select Application page, click **Select app** in the Think Application window.

The screenshot shows the 'Select Application' window. At the top, it says 'Select Application' and 'Global Knowledge®'. Below this is a large grey 'GK' logo. In the center, there is a card for the 'Think Application'. It features a pink circular icon with a smartphone and gear. The text 'Think Application' is displayed above a 'Description' section. The description reads: 'A web application to browse items for sale, and to submit reviews.' At the bottom of the card is a blue 'Select App' button.

- \_\_\_ g. The confirm subscription page is displayed.

## Confirm Subscription

A subscription will be created for the product, plan and application you selected.

### Product

think-product

### Plan

Default Plan

### Application

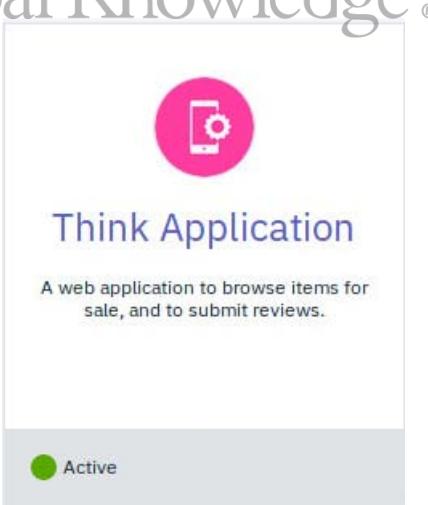
Think Application

Previous

Next

Click **Next**.

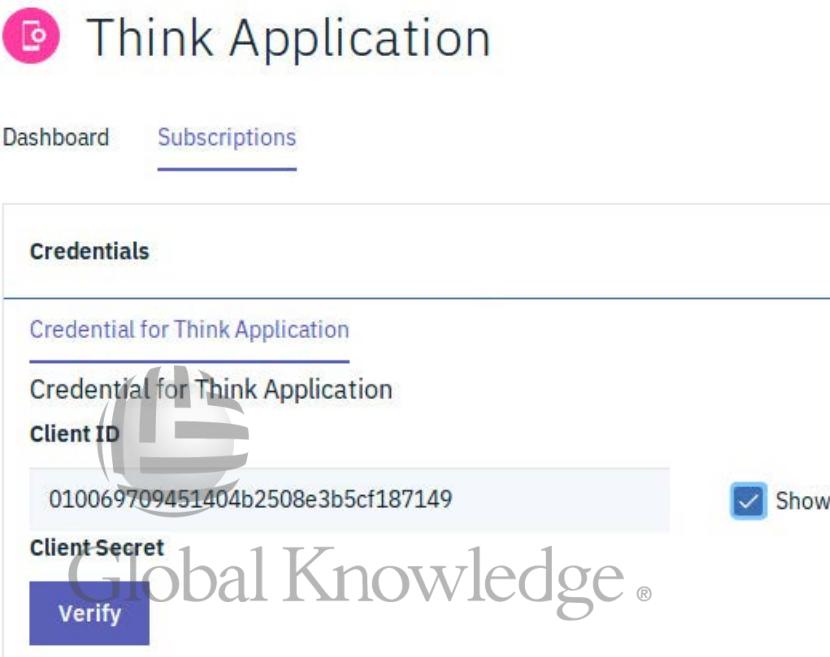
- \_\_\_ h. The subscription is complete. Click **Done**.
- \_\_\_ 2. Validate the client secret for the think application.
- \_\_\_ a. Click the **Apps** link in the Developer Portal menu.
- \_\_\_ b. Click the **Think Application** link.



- \_\_\_ c. Click the Subscriptions tab. Notice that the application now includes a subscription to the think-product and default plan.

| Subscriptions         |              |   |
|-----------------------|--------------|---|
| PRODUCT               | PLAN         |   |
| think-product (1.0.0) | Default Plan | ⋮ |

- \_\_\_ d. From the subscriptions page, click the **Verify** icon for the client secret.



The screenshot shows the Think Application dashboard with the Subscriptions tab selected. Under the Credentials section, there is a credential for the Think Application. The Client ID is listed as 010069709451404b2508e3b5cf187149. Below it, the Client Secret field contains a placeholder value: .....|. To the right of the Client Secret field is a blue 'Verify' button. To the right of the 'Verify' button is a checked checkbox labeled 'Show'.

- \_\_\_ e. Paste the client secret that you saved into the verify dialog.

## Verify an application Client Secret

Use this form to verify you have the correct client secret for this application.

**Secret \***

Click **Submit**.

- \_\_\_ f. The client secret should verify successfully.



## Important

If the validation fails, the client secret was not saved correctly.

🚫 Validation of the credential failed. ✖

Applications

⌚ Think Application ⋮

You must delete the Think Application and re-create it. Then, resave the client ID and client secret.



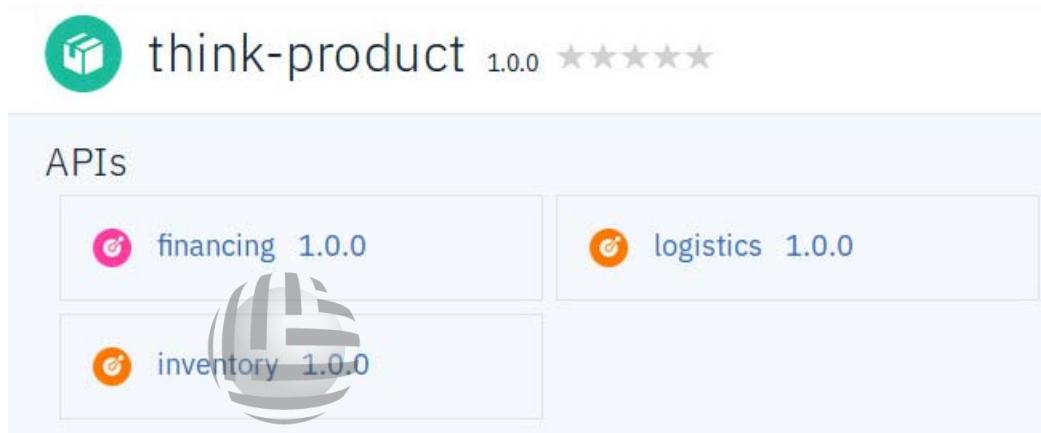
## 11.4. Test subscribed APIs in the Developer Portal test client

In the last section, you subscribed the Think application to the default plan from the think-product.

In this section, test two APIs from the think-product in the Developer Portal test client. Developers browse and test APIs on the Developer Portal before they call them from their applications.

Examine and test the GET /calculate operation from the financing API. Test the GET /stores operations from the logistics API.

- 1. Test the GET /calculate operation from the financing API definition.
  - a. The think-product is open on the page.
  - b. Click **financing** in the list of APIs.



- \_\_\_ c. The Developer Portal test feature page opens. The overview page for the financing API is displayed.

The screenshot shows the IBM Developer Portal interface. At the top, there is a search bar with a magnifying glass icon and the word "financing". To the right of the search bar is a blue button with three vertical dots. Below the search bar, the API name "financing" is displayed in large blue text, followed by the version "1.0.0" and a five-star rating. On the left side, there is a sidebar with a "Filter" button and two menu items: "Overview" (which is currently selected and highlighted in blue) and "Definitions". Under "Overview", there is a link to "GET /calculate". The main content area has a title "Overview" and several sections: "Download" with options "Open API Document", "Type REST", "Endpoint Production, Development: https://apigw.think.ibm/think /sandbox/financing", and a "Security" section. The security section includes a "clientIdHeader" field with the value "X-IBM-Client-Id" and a note that it is an apiKey located in header.

- \_\_\_ d. Select GET /calculate from the left column.
- \_\_\_ e. The details of the operation and parameters are displayed. You see that the endpoint for the GET operation is a location on the API gateway.

\_\_\_ f. Click the **Try it** tab in the right column.

## get.financingAmount

Details      [Try it](#)

**GET**      **Production, Development:** <https://apigw.think.ibm/think/sandbox/financing/calculate>

Security

**Identification**

**Client ID**

Think Application ▾

\_\_\_ g. The Developer Portal automatically inserts the client ID from the Think Application.



- \_\_ h. Scroll down to the parameters area on the page. Type the following values into the test client:
- Amount: 300
  - Duration: 24
  - Rate: 0.04

The screenshot shows a REST API test client interface. At the top, there's a dropdown menu labeled "Parameter" with "Header" selected. Below it, under "Accept", is a dropdown set to "application/json". A large "Query" section is expanded, containing three fields: "amount\*", "duration\*", and "rate\*". Each field has a text input box with a numeric value and up/down arrow buttons for adjustment. To the right of each input box is a "Generate" link. At the bottom of the "Query" section are two buttons: "Reset" and "Send". The background features a watermark of the Global Knowledge logo and the text "Global Knowledge".

| Parameter | Value |
|-----------|-------|
| amount*   | 300   |
| duration* | 24    |
| rate*     | 0.04  |

Click **Send**.

- \_\_\_ i. Confirm that the operation returned a status code of 200 OK.

Response

```

Code: 200 OK
Headers:
content-type: application/json
x-ratelim-limit: name=default,100;
x-ratelim-remaining: name=default,99;
{
 "soapenv:Envelope": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "soapenv:Body": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "ser:financingResult": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "ser:paymentAmount": {

```



- \_\_\_ j. Scroll down in the response area to see the calculated payment amount.

Global Knowledge ®

```

 "ser:paymentAmount": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "$": "12.51"
 }
 }
}

```



## Questions

How did the financing API return a successful response even though the inventory application is not running?

Answer: Although the financing API is part of the same think-product, the API operation is called independently from the inventory application. The financing API calls a SOAP service that runs on the Services domain of the DataPower gateway. Essentially, the financing API uses a different back-end to the inventory API.

- \_\_\_ k. Click the API Products tab in the Developer Portal menu to display the products and APIs.

- \_\_\_ I. Click the link in the think-product to display the list of APIs.
- \_\_\_ 2. Test the operations from the logistics API definition.
  - \_\_\_ a. The think-product is open on the page.
  - \_\_\_ b. Click **logistics** in the list of APIs.

The screenshot shows a product page titled "think-product 1.0.0" with a 5-star rating. Below the title, it says "APIs". There are three service entries:

- "financing 1.0.0" with a pink icon
- "logistics 1.0.0" with an orange icon
- "inventory 1.0.0" with an orange icon

- \_\_\_ c. In the **think 1.0.0** product, select the **logistics** entry.



- \_\_ d. The overview page for the logistics API is displayed.

The screenshot shows the API documentation for the logistics API. At the top, there is a logo with a gear icon and the word "logistics" next to it, followed by the version "1.0.0" and five stars. Below the header, there is a sidebar with "Overview" selected, showing links for "GET /shipping" and "GET /stores". The main content area has a title "Overview" and a "Download" button with a PDF icon. It also includes "Open API Document" and "Type REST". Under "Endpoint", it lists "Production, Development" and the URL "https://apigw.think.ibm/think/sandbox/logistics". In the "Security" section, it shows "clientID" and "X-IBM-Client-Id" with the note "apiKey located in header".

Global Knowledge ®

The logistics API includes two operations, `GET /shipping` and `GET /stores`. To reduce the duration of the policies exercise, the `GET /shipping` operation was not completed. The GET shipping branch in the assembly simply flowed through the sequence. If you decide to test the GET shipping operation with a valid US postal code in the Developer Portal, you get a 200 OK response with nothing else. The exercise did not implement any policies for the `GET /shipping` operation.

**Request**

```
GET https://apigw.think.ibm/think/sandbox/logistics/shipping?zip=98121
Headers:
Accept: application/json
X-IBM-Client-Id: 010069709451404b2508e3b5cf187149
```

**Response**

```
Code: 200 OK
Headers:
x-global-transaction-id: 196c55555c78865400000433
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,98;
```

- \_\_\_ 3. Test the `GET /stores` operation in the logistics API.
  - \_\_\_ a. Select the `GET /stores` operation from the left column.
  - \_\_\_ b. In the right column, click the **Try it** tab.

- \_\_\_ c. Type 15222 in the **zip** input parameter.

The screenshot shows a configuration interface for a POST request. It includes sections for Header, Accept, Content-Type, and Query. The Query section contains a parameter named 'zip' with the value '15222'. Below the form are 'Reset' and 'Send' buttons.

| Parameter    | Header                                            |
|--------------|---------------------------------------------------|
| Accept       | application/json                                  |
| Content-Type | application/json                                  |
| Query        | <b>zip*</b><br><input type="text" value="15222"/> |

Generate

**Reset**    **Send**

Click **Send**.

- \_\_\_ d. Confirm that the operation returns a status of 200 OK.

The screenshot shows a REST API testing interface. On the left, under 'Request', it lists operations: GET /shipping, GET /stores (selected), and Definitions. On the right, under 'Response', it shows the API endpoint, headers, and a JSON response body.

**Request**

- GET /shipping
- GET /stores**
- Definitions

**Global Knowledge**

GET https://apigw.think.ibm/think/sandbox/logistics/stores?zip=15222  
 Headers:  
 Accept: application/json  
 X-IBM-Client-Id: 010069709451404b2508e3b5cf187149

**Response**

```

Code: 200 OK
Headers:
content-type: text/xml
x-global-transaction-id: 196c55655c788b6700000242
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,97;
{"maps_link":"https://www.openstreetmap.org/#map=16/40.44048745/-79.9990337281745"}

```



## Questions

How did the logistics API return a successful response even though the inventory application is not running?

Answer: The logistics API calls an external service that runs on the website [openstreetmap.org](http://openstreetmap.org).

- 4. Sign out of the Developer Portal.
- 5. Close the web browser.
- 6. In the next part, you update the web-based client application with the client ID and secret. Then, you test all the APIs of the think-product from the web-based client application.



Global Knowledge ®

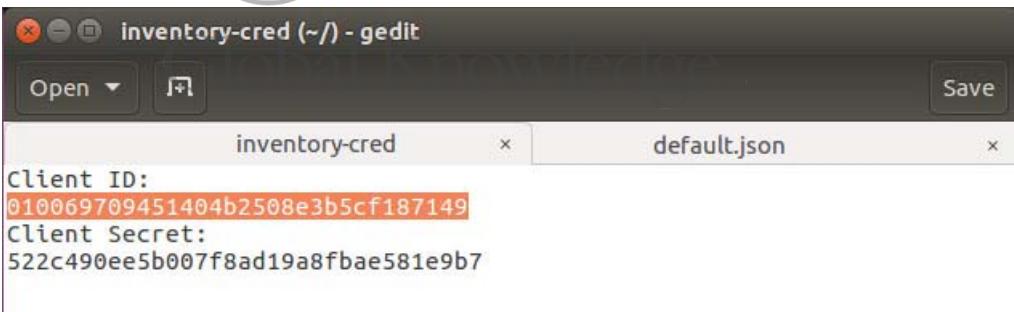
## 11.5. Update the client application with the client ID and client secret

In a previous exercise, you defined two security requirements on the `inventory` API definition:

- All operations require an `API key`. The client must provide a valid `client ID` and `client secret` value in the HTTP request header when it calls an API operation.
- All `inventory` operations require OAuth authentication. In a two-step process, the client application must retrieve a valid authorization code from the OAuth Provider. After authorization, the client must exchange the code for an access token. The client must send the access token with every API operation call.

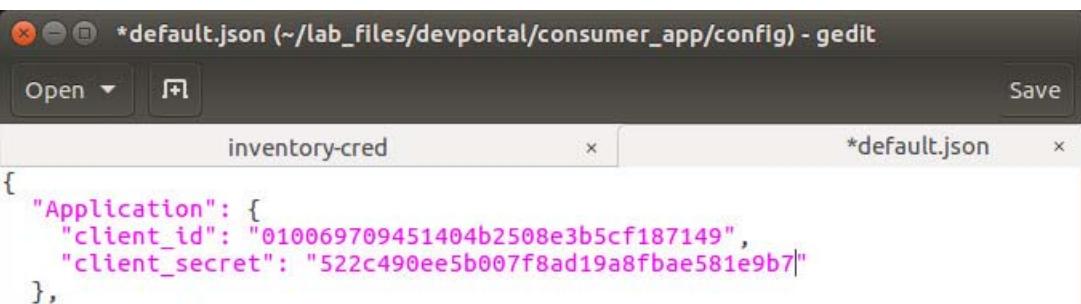
In this section, you add the `client ID` and `client secret` values to a configuration file in a sample JavaScript client application. You test the `GET /inventory/items` API operation with the web client that implements an oauth authorization and token exchange.

- 1. Open the files in File Manager.
  - a. From the home page, select the `inventory-cred` file into which you saved the client ID and client secret values. Right-click the file. Then, select Open with gedit. If you minimized the gedit file earlier, you can maximize it instead of opening the file in the editor.
  - b. In the File Manager, navigate to the `~/lab_files/devportal/consumer_app/config` directory. Right-click the `default.json` file, then click **Open with gedit**.
  - c. The editor is open with two tabs, one for each of the two files.



```
inventory-cred (~/) - gedit
Open Save
inventory-cred default.json
Client ID:
010069709451404b2508e3b5cf187149
Client Secret:
522c490ee5b007f8ad19a8fbe581e9b7
```

- 2. Copy the `client ID` and `client secret` values from the saved file into the consumer application.
  - a. Copy and paste the client ID and client secret values from the `inventory-cred` file to the `default.json` configuration file.



```
*default.json (~/lab_files/devportal/consumer_app/config) - gedit
Open Save
inventory-cred *default.json
{
 "Application": {
 "client_id": "010069709451404b2508e3b5cf187149",
 "client_secret": "522c490ee5b007f8ad19a8fbe581e9b7"
 }
}
```

- \_\_\_ b. Review the values in the default.json file.

```

"API-Server": {
 "protocol": "https",
 "host": "apigw.think.ibm",
 "org": "think",
 "catalog": "sandbox"
},
"APIs": {
 "inventory": {
 "base_path": "/inventory",
 "require": [
 "client_id",
 "client_secret",
 "oauth"
]
 },
 "financing": {
 "base_path": "/financing",
 "require": [
 "client_id"
]
 },
 "logistics": {
 "base_path": "/logistics",
 "require": [
 "client_id"
]
 },
 "oauth20": {
 "base_path": "/oauth20",
 "paths": {
 "authz": "/oauth2/authorize",
 "token": "/oauth2/token"
 },
 "grant_types": [
 "password"
],
 "scopes": [
 "inventory"
]
 }
}

```



These values match the paths that were set for the APIs in earlier exercises.

- \_\_\_ c. Save the changes to the `default.json` file.



## 11.6. Test all the APIs with the consumer client web application

In this section, you test all the APIs in the think-product from the consumer client web application. The consumer client web application runs an integrated test of all the APIs in the exercise case study.

- 1. Start the Loopback application in the Docker runtime environment.
  - a. Open a terminal window.
  - b. Start the Docker image that runs the inventory API.

```
$ docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image
```

```
> inventory@1.0.0 start /inventory
> node .
```

```
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```



### Note



You test the client application with a single instance of the inventory application that runs in the Docker runtime. This course does not cover the integration of the Docker image into the Kubernetes runtime.

- 2. Start the consumer application from another terminal application.

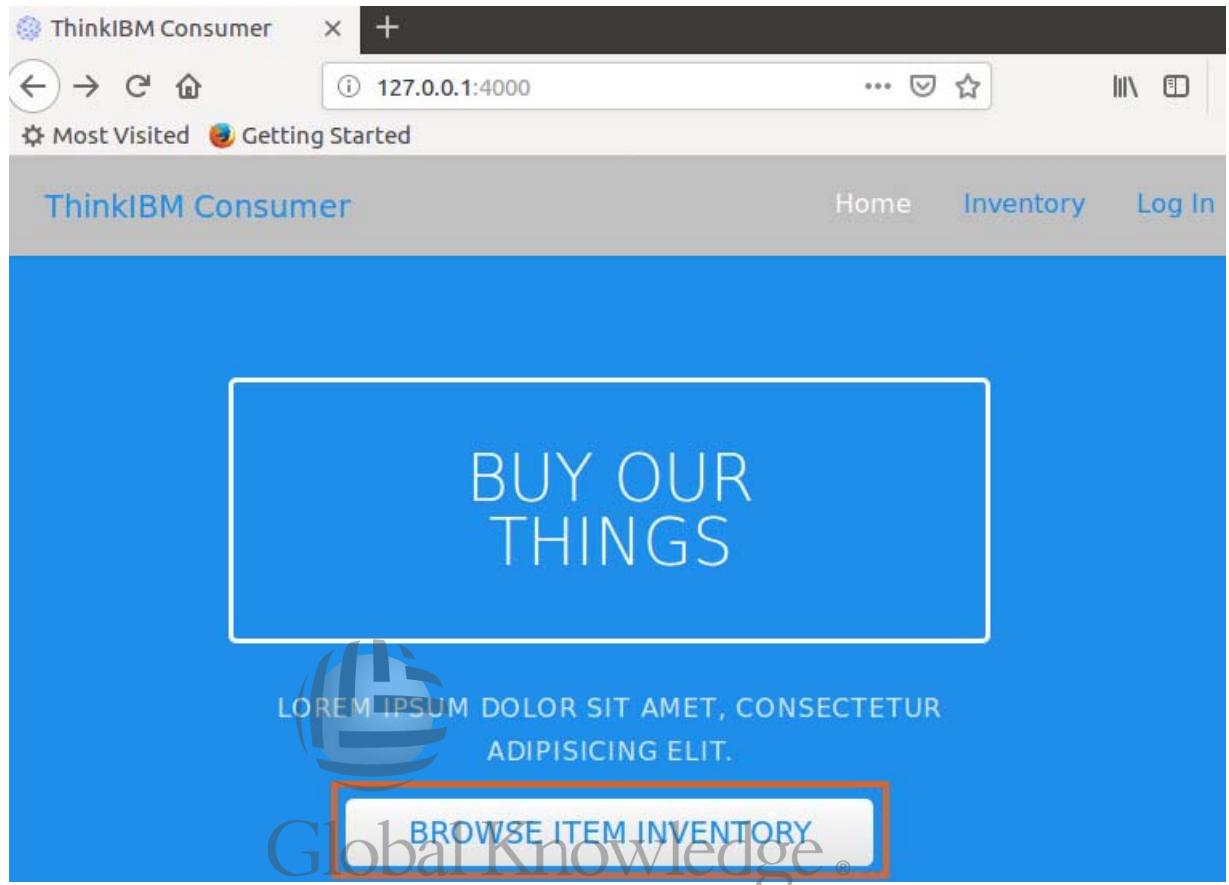
- a. Open the Terminal with **Terminal > New Terminal** from the Terminal menu.
- b. Navigate to the `consumer_app` directory.

```
$ cd ~/lab_files/devportal/consumer_app/
$ pwd
/home/localuser/lab_files/devportal/consumer_app/
```

- c. Start the consumer application.

```
$ npm start
```

- \_\_\_ 3. Test the `GET /inventory/items` API operation from the consumer application.
- \_\_\_ a. When you start the consumer application, a web browser opens to the main page. If the consumer application does not start automatically, type `127.0.0.1:4000` in the address of a browser window.



Click **BROWSE ITEM INVENTORY**, or click **Log In**.

- \_\_\_ b. Type:
- User name: `user`
  - Password: `pass`

Click **Log In**.

**Note**

The oauth service accepts any non-blank values for the user name and password values.

- \_\_\_ c. The page displays the list of inventory items that are retrieved from the mySQL database.

The screenshot shows a web browser window titled "ThinkIBM Consumer" with the URL "127.0.0.1:4000/inventory". The page displays three inventory items:

- Calculator**: An image of a vintage-style calculator unit with a keyboard and a small screen. Price: \$5199.99, Rating: 5 stars.
- Collator**: An image of a dark-colored industrial collator machine. Price: \$2799.99, Rating: 5 stars.
- Computing Scale**: An image of a mechanical computing scale. Price: \$699.99, Rating: 5 stars.

**Information**

The user login page is part of the OAuth 2.0 authentication message flow. In the first step, a user selects the browse item inventory or log in link on the main page. Since the GET /inventory/items operation is secured with OAuth authentication, a valid resource owner must log in and grant permission to access the /inventory/items resource.

You can view the output in the terminal emulator to see the output from the OAuth message flow:

- 1) The user requests access to the /inventory/items resource.

```
GET /inventory 302 10.850 ms - 56
GET /login 200 124.067 ms - 1622
```

- 2) The consumer application redirects the request to an OAuth login page. The resource owners enter their user name and password. The client application sends the credentials to the

<https://apigw.think.ibm/think/sandbox/oauth20/oauth2/authorize> API operation to verify the credentials.

Initializing OAuth client...

Authorization URL:

<https://apigw.think.ibm/think/sandbox/oauth20/oauth2/authorize>

|                  |                                                                                                                                     |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Token URL:       | <a href="https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token">https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token</a> |
| Client ID:       | a4f7032eaeb198f2b95b646415cal1a08                                                                                                   |
| Client Secret:   | ada6fc205787977db818c529301be5c2                                                                                                    |
| Auth Grant Type: | password                                                                                                                            |
| Redirect URI:    | (undefined)                                                                                                                         |
| Scope:           | inventory                                                                                                                           |

- 3) The sample authentication service accepts the credentials. In turn, the /oauth2/authorize API operation returns an authorization code to the client application.
- 4) The client application exchanges the authorization code for an **access token** with a call to <https://apigw.think.ibm/think/sandbox/oauth20//oauth2/token>. The token service returns a time-limited token to the client application.

... authorize and token calls successful.

Using Access Token:

```
AAIgYTRmNzAzMmVhZWIxOThmMmI5NWI2NDY0MTVjYTFhMDjdulPPXtvTXccRF5-jgJ2kMTPZ1cJSh_6
-VAiIakQnriWI91AJwegZ_eP_WgZy_X5j6jJLTuSTIXW3nOwSRddx
POST /login 302 153.274 ms - 64
```

- 5) On subsequent calls to /inventory/items, the client application sends the access token in the HTTP request message header.

MY OPTIONS:

```
{ "method" : "GET" , "url" : "https://apigw.think.ibm/think/sandbox/inventory/items?filter[order]=name ASC" , "strictSSL":false , "headers" : { "X-IBM-Client-Id": "a4f7032eaeb198f2b95b646415cal1a08" , "X-IBM-Client-Secret": "ada6fc205787977db818c529301be5c2" , "Authorization": "Bearer AAIgYTRmNzAzMmVhZWIxOThmMmI5NWI2NDY0MTVjYTFhMDjdulPPXtvTXccRF5-jgJ2kMTPZ1cJSh_6-VAiIakQnriWI91AJwegZ_eP_WgZy_X5j6jJLTuSTIXW3nOwSRddx" } }
GET /inventory 200 296.467 ms - 9823
```

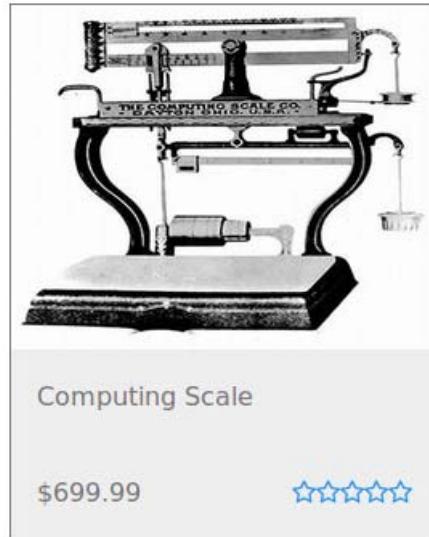
- 
- 4. Confirm that the client application displays inventory items in the page. Next, you test the other features of the case study from the consumer client application.



## Troubleshooting

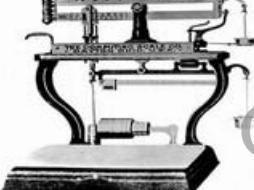
If you do not see a list of inventory items, review [Exercise 12, "Troubleshooting the case study - optional"](#) for instructions on how to correct configuration issues in your API Connect Cloud.

- \_\_\_ 5. Test the review function in the consumer client application.  
\_\_\_ a. Click the Computing Scale image on the web page.



- \_\_\_ b. The details for the Computing Scale are displayed on the page.

ThinkIBM Consumer      Home      Inventory      Log in

 **Computing Scale**   
**Global Knowledge** ® 

**Calculate Shipping**   
**Calculate** [Calculate Monthly Payment](#)

---

### Product Description

In 1885 Julius Pitrat of Gallipolis, Ohio, patented the first computing scale. Six years later, Edw Orange Ozias of Dayton, Ohio, purchased Pitrat's patents and incorporated The Computing S as the world's first computing scale vendor. And four years after that, The Computing Scale C introduced the first automatic computing scale, shown here. In 1911, the Computing Scale C merged with the International Time Recording Company and Tabulating Machine Company to Computing-Tabulating-Recording Company, a business that was renamed IBM in 1924.

- \_\_\_ c. Scroll down to the Customer Reviews for the Computing Scale. No reviews currently exist.

### Customer Reviews (0)

### Write a review

A form titled "Write a review" with the following fields:

- Rating: A dropdown menu showing the value "5".
- Name: An input field containing "jdoe@example.com".
- Comment: An input field containing "Comment".
- Submit: A blue button labeled "Submit".

- \_\_\_ d. Select a rating and write a review.

A form titled "Write a review" with the following fields:

- Rating: A dropdown menu showing the value "1".
- Name: An input field containing "kevin@think.ibm".
- Comment: An input field containing "So, so".
- Submit: A blue button labeled "Submit".

The background features a watermark of the Global Knowledge logo and the text "Global Knowledge ®".

Click **Submit**.

- \_\_\_ e. The review is added to the bottom of the page.

### Customer Reviews (1)

★★★★★ by Kevin on Sat Mar 02 2019  
"So, so"

- \_\_\_ f. The average of all the reviews is calculated and displayed at the top of the page for the item.

**Computing Scale**

★★★★★

\$699.99

[Calculate Monthly Payment](#)

- \_\_\_ g. Notice that some entries are also added to the log in the terminal window.

DATA:

```
{ "date": "2019-03-02T01:46:14.594Z", "reviewer_name": "Kevin", "reviewer_email": "kevin@think.ibm", "comment": "So, so", "rating": 1, "id": "5c79e066781e9300190ada44", "itemId": 3}
```

POST /item/submitReview 302 77.964 ms - 58

GET /item/3 200 388.469 ms - 4729

- \_\_\_ 6. Test the financial API from the consumer web application.

- \_\_\_ a. From the details page of one of the inventory items, select **Calculate Monthly Payment**.

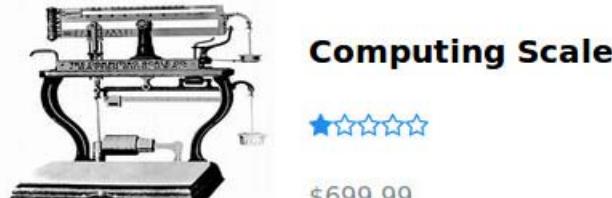
**Computing Scale**

★★★★★

\$699.99

[Calculate Monthly Payment](#)

- \_\_\_ b. The financial API returns the calculated payment.



Monthly Payment: \$30.37 at 3.9% for 24 months

- \_\_\_ c. Notice that an entry is also added to the log in the terminal window.

```
GET /financing/calculate/699.99 200 97.938 ms - 493
```

- \_\_\_ 7. Test the logistics API from the consumer web application.

- \_\_\_ a. From the details page of one of the inventory items, go to the Calculate Shipping area on the page.

The screenshot shows a product detail page for a "Computing Scale". It includes an image of the scale, the product name, a five-star rating, and the price "\$699.99". Below the product information, it says "Monthly Payment: \$30.37 at 3.9% for 24 months". To the right, there is a "Calculate Shipping" section with a "Zip Code" input field and a "Calculate" button.

- \_\_\_ b. Type in a valid US postal code.

**Calculate  
Shipping**

15222

**Calculate**

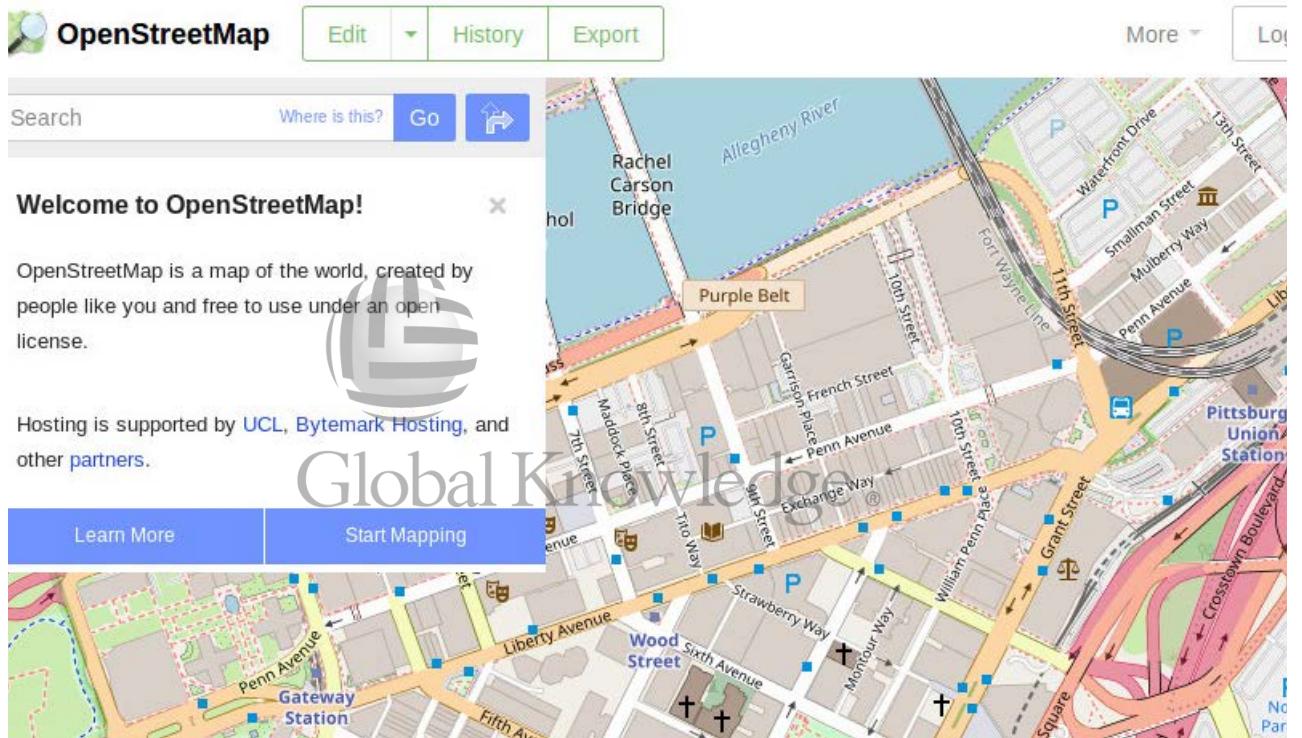
Click **Calculate**.

- c. Recall that the logistics GET /shipping function was not completed in this course. No shipping costs are shown.

## Calculate Shipping

Pickup from nearest  
store

- d. Click **Pickup from nearest store** to call the GET /stores operation from the logistics API.
- e. The OpenStreetMap for the selected US postal code is displayed.



- f. The call operations are displayed in the terminal window.

```
GET /logistics/shipping/15222 200 22.660 ms - -
GET /logistics/stores/15222 200 715.862 ms - 87
```

8. You have successfully completed the course case study.

## End of exercise

## Exercise review and wrap-up

The first part of the exercise covered the client application registration process in the Developer Portal. To use APIs that are published in API Connect, the application developer must create an application and subscribe to API. The Developer Portal issues a client ID and client secret. These credentials uniquely identify a client application when it calls API operations.

The second part of the exercise is focused on testing the operations of the think-product and the APIs in the case study. In a previous exercise in this course, you built a LoopBack API application that returns item records from the inventory database. In subsequent exercises, you secured the API with OAuth 2.0 authorization. You configured a client web application with the client credentials and tested all the APIs that you created in the last step.



# Exercise 12. Troubleshooting the case study - optional

## Estimated time

00:30

## Overview

In this exercise, you review and apply troubleshooting steps in the exercise case study. You apply the steps to verify the operation of the inventory application and correct the issues that you identify. This exercise is only required if the student is not getting the correct results.

## Objectives

After completing this exercise, you should be able to:

- Verify the key components in the exercise case study
- Troubleshoot and correct common issues with the case study
- Troubleshoot general issues when running through the course exercises
- Useful Kubernetes information and commands

## Introduction

Global Knowledge®

In the previous exercise, you tested the logistics, financing, and inventory APIs in the Developer Portal test client. You also tested the OAuth 2.0 message flow with the consumer app, a client-side JavaScript client.

## Requirements

Before you start this exercise, you must complete the data sources, remote, policies, authorization, containers, publish, and developer portal exercises in this course.

You must complete this lab exercise in the student development workstation: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.

## Exercise instructions



Global Knowledge®

## 12.1. Review the inventory items API operation

In the developer portal exercise, you used the consumer client application to test the GET /inventory/items operation. If you encounter a runtime error when you test the operation, you must determine which part of the API operation caused the issue.

```
localuser@ubuntu: ~/lab_files/devportal/consumer_app
GET /stylesheets/pure/layouts/login.css 304 1.218 ms -
GET /javascripts/calc-shipping.js 304 1.226 ms -

Initializing OAuth client...
Authorization URL: https://apigw.think.ibm/think/sandbox/oauth20/oauth2/authorize
Token URL: https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token
Client ID: 4011e4b6775fbf1728c6977ed0ea52d3
Client Secret: cd65fb95535f9b809cda8c3a68566a66
Auth Grant Type: password
Redirect URI: (undefined)
Scope: inventory

... authorize and token calls successful.
Using Access Token: AAIgNDAxMWU0YjY3NzVmYmYxNzI4YzY5NzdlZDBlYTUyZDPSW7f434PL_AA
IruJ-Yuo2aEdnaVE0xZHId1Uqoos-cQNMU0ladff79nn3GtyfArcjhLZmksFpcPTKfv4lFGy0
POST /login 302 125.925 ms - 64
MY OPTIONS:
{"method": "GET", "url": "https://apigw.think.ibm/think/sandbox/inventory/items?filter[order]=name ASC", "strictSSL": false, "headers": {"X-IBM-Client-Id": "4011e4b6775fbf1728c6977ed0ea52d3", "X-IBM-Client-Secret": "cd65fb95535f9b809cda8c3a68566a66", "Authorization": "Bearer AAIgNDAxMWU0YjY3NzVmYmYxNzI4YzY5NzdlZDBlYTUyZDPSW7f434PL_AA"}, "body": null}
```

The consumer\_app makes 3 API calls when you click Log in:

- Call #1: Consumer app > API gateway /oauth2/authorize operation
- Call #2: Consumer app > API gateway /oauth2/token operation
- Call #3: Consumer app > API gateway /inventory/items > > **Inventory LoopBack API**

In *call #1*, the consumer application calls the /oauth2/authorize operation from the OAuth 2.0 Provider API. The authorize operation redirects your web browser to a login page. You enter any value for the user name and password, and the authorization URL returns HTTP status code 200 to the OAuth 2.0 Provider. In turn, the authorize API operation returns an OAuth authorization bearer token to the consumer application.

In *call #2*, the consumer application sends the authorization bearer token to the /oauth2/token operation at the OAuth 2.0 Provider API. After the token API operation verifies that the bearer token is valid, it returns an OAuth 2.0 access token to the consumer application.

In *call #3*, the consumer application calls the /inventory/items API operation with the access token.

## 12.2. Test the OAuth authorize and token API operations

When you call an operation in the inventory API, OAuth 2.0 Provider must authenticate the resource owner identity and authorize your request.

The OAuth 2.0 message flow consists of two steps:

- In step 1, the consumer app sends the resource owner user name and password to the /oauth2/authorize operation. If the user is authenticated, the operation returns an authorization bearer token.
- In step 2, the consumer app sends the authorization bearer token to the /oauth2/token operation. The token operation exchanges a valid bearer token for an access token.

In this section, make sure that the **authorize** and **token** API operations are hosted at the API gateway. Test the operations before you proceed to the next section.

— 1. Test the /oauth2/authorize and /oauth2/token operations with the consumer application.

— a. Open a terminal window.

— b. Change directory to the consumer client application, at:

`~/lab_files/devportal/consumer_app`

`$ cd ~/lab_files/devportal/consumer_app`

`$ pwd`

`/home/localuser/lab_files/devportal/consumer_app`

— c. Start the consumer app.

`$ npm start`

— d. Confirm that the consumer app started successfully.

```
> consumer@0.0.0 start /home/localuser/lab_files/devportal/consumer_app
> node ./bin/www
```

```
GET / 304 666.936 ms --
```

```
GET /stylesheets/pure/pure-min.css 304 10.010 ms --
```

```
GET /stylesheets/pure/grids-responsive-min.css 304 9.816 ms --
```

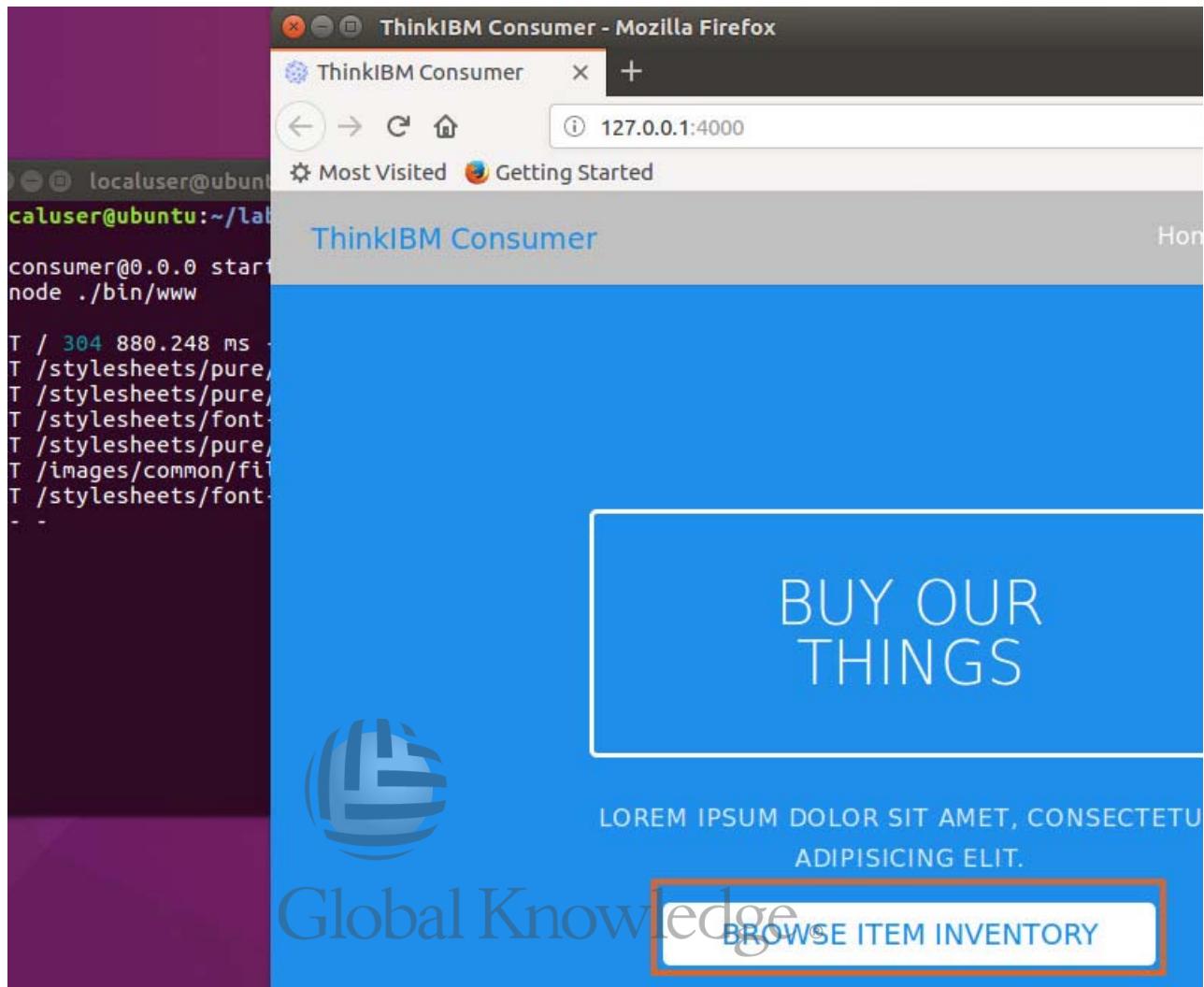
```
GET /stylesheets/font-awesome/font-awesome.css 304 11.010 ms --
```

```
GET /stylesheets/pure/layouts/main.css 304 13.720 ms --
```

```
GET /images/common/file-icons.png 304 0.344 ms --
```

```
GET /stylesheets/font-awesome/fonts/fontawesome-webfont.woff?v=4.0.3 304
3.944 ms --
```

- \_\_ e. In the consumer application web page, click **BROWSE ITEM INVENTORY**.



- \_\_\_ f. In the resource owner login page, enter the following values:

- User name: **user**
- Password: **pass**

The screenshot shows a web browser window titled "ThinkIBM Consumer". The address bar displays the URL "127.0.0.1:4000/login". Below the address bar, there are links for "Most Visited" and "Getting Started". The main content area is titled "ThinkIBM Consumer". It contains two input fields: one for "Username" and one for "Password", both currently empty. Below these fields is a large blue "Log In" button.

Click **Log in**.

- \_\_\_ 2. Review the results from the **authorize** and **token** API calls.

- \_\_\_ a. Review the console log entries in the terminal window.

```
Initializing OAuth client...
Authorization URL:
https://apigw.think.ibm/think/sandbox/oauth20/oauth2/authorize
Token URL:
https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token
Client ID: a4f7032eaeb198f2b95b646415ca1a08
Client Secret: ada6fc205787977db818c529301be5c2
Auth Grant Type: password
Redirect URI: (undefined)
Scope: inventory
```

## Information

The console displays the configuration settings and the results from calling the authorize and token services at the OAuth 2.0 Provider API.

- The **authorize path** and **token path** must match the values that you defined in the configuration of the Native OAuth Provider definition.
- The **client ID** and **client secret** values must match the values that you verified in the Developer Portal.
- The **authorization grant type** is set to **password**.
- The **scope** is set to **inventory**.

- The redirect URI is not specified.

If you sent the correct client ID and secret values to the authorize operation, the server sends back an authorization bearer token. The consumer application calls the token operation to exchange the bearer token for an access token.

---



## Troubleshooting

If you see a `"cannot retrieve OAuth access token"` error message, the **OAuth provider API** rejected the login attempt from the consumer application.

`... cannot retrieve OAuth access token.`

Reason:

```
{ "status":401,"body":{ "error":"unauthorized_client", "error_description":"Invalid Client ID or secret, or client not subscribed to this API"}}
```

```
localuser@ubuntu: ~/lab_files/devportal/consumer_app

Initializing OAuth client...
Authorization URL: https://apigw.think.ibm/think/sandbox/oauth20/oauth2/authorize
Token URL: https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token
Client ID: a4f7032eaeb198f2b95b646415ca1a08
Client Secret: ada6fc205787977db818c529301be5c2
Auth Grant Type: password
Redirect URI: (undefined)
Scope: inventory

... cannot retrieve OAuth access token.
Reason: {"status":401,"body":{ "error":"unauthorized_client", "error_description":"Invalid Client ID or secret, or client not subscribed to this API"}}
```

Two scenarios can cause this error:

- If the **client ID** and **client secret** values do not match the values in the Developer Portal, the **authorize** operation rejects the request. Verify the credentials for the client application again.

- If the **client ID** and **client secret** values *do match* the Developer Portal settings, make sure that you subscribed the application to the `think-product` in the Portal.

The screenshot shows the IBM Developer Portal interface. At the top, there's a navigation bar with a logo, the text "Think Application", and three vertical dots. Below the navigation, there are two tabs: "Dashboard" and "Subscriptions", with "Subscriptions" being the active tab. Under the "Subscriptions" tab, there's a section titled "Credentials" which contains fields for "Client ID" and "Client Secret". The "Client ID" field is filled with a long string of characters and has a "Show" link next to it. The "Client Secret" field is empty. Below the credentials is a blue "Verify" button. In the background, there's another panel titled "Subscriptions" showing a list of products and plans. One entry, "think-product (1.0.0)", is highlighted with a red border. This entry has columns for "PRODUCT" and "PLAN", both of which are "think-product (1.0.0)". A "Default Plan" link is also visible. The overall theme of the portal is light blue and white.

- If the client ID and secret match the values in the Developer Portal and the application is subscribed to the `think-product`, then this issue might be the result of restarting the student workstation. In this case, delete the Think Application and re-create it. Use the newly generated client ID and client secrets.



## Troubleshooting

If you see an **"API not found for requested URI"** error message, the consumer application cannot reach the `/oauth2/authorize` or `/oauth2/token` operations. Verify the **base path** and **path** settings in the **Native OAuth Provider** definition.

... cannot retrieve OAuth access token.

```
Reason: {"status":404,"body":{ "httpCode": "404" , "httpMessage": "Not Found" , "moreInformation": "API not found for requested URI" }}
```



## Important

If you see a **Using Access Token** message, you configured the OAuth authorization correctly.

... authorize and token calls successful.

Using Access Token:

AAEkMGNkN2ZmNzMtMjAyNi00ZDU0LTk2YTEtNTA4NmQ5M2FkODc2M37UsgOHiltIvtMl7\_wp3oXm23ZhLQSCs5ZA05n7QgwGxbcozbdGmY2V4oUvnznkR9fxnjLbZ8IU3dwNqJ-4hQ

You can review the following exercise sections, or you can skip ahead to [Section 12.4, "Test the inventory items API operation,"](#) on page 12-11.



Global Knowledge®

## 12.3. Verify the OAuth 2.0 Provider API configuration

You can verify the OAuth 2.0 Provider API configuration with the API Manager web application. Refer to [Section 8.4, "Test OAuth security in the inventory API,"](#) on page 8-20.



## 12.4. Test the inventory items API operation

In the last section of the troubleshooting exercise, you confirmed that the **authorize** and **token** operations ran successfully. The consumer application is authorized to call any operation in the inventory API.

In the terminal window, the application log displays an attempt to call the GET /inventory/items operation.

```

Initializing OAuth client...
Authorization URL:
https://apigw.think.ibm/think/sandbox/oauth20/oauth2/authorize
Token URL: https://apigw.think.ibm/think/sandbox/oauth20/oauth2/token
Client ID: 4011e4b6775fbf1728c6977ed0ea52d3
Client Secret: cd65fb95535f9b809cda8c3a68566a66
Auth Grant Type: password
Redirect URI: (undefined)
Scope: inventory

... authorize and token calls successful.

Using Access Token:
AAIgNDAxMWU0YjY3NzVmYmYxNzI4YzY5Nzd1ZDB1YTUyZDMe0RuvIQIrc1NlHrP0X7_Do3Tet jvj9hE
U69Jz1xEfNOhOceWkCeof9eDKQJIkh116upwpP32635TtHOJ_d1pu
POST /login 302 166.728 ms - 64

MY OPTIONS:
{
 "method": "GET",
 "url": "https://apigw.think.ibm/think/sandbox/inventory/items?filter[order]=name%20ASC",
 "strictSSL": false,
 "headers": {
 "X-IBM-Client-Id": "4011e4b6775fbf1728c6977ed0ea52d3",
 "X-IBM-Client-Secret": "cd65fb95535f9b809cda8c3a68566a66",
 "Authorization": "Bearer AAIgNDAxMWU0YjY3NzVmYmYxNzI4YzY5Nzd1ZDB1YTUyZDMe0RuvIQIrc1NlHrP0X7_Do3Tet jvj9hE
U69Jz1xEfNOhOceWkCeof9eDKQJIkh116upwpP32635TtHOJ_d1pu"
}

```

After the consumer application receives an **access token**, it sends an HTTP GET request to the /inventory/items operation. The full path for the API operation is <https://apigw.think.ibm/think/sandbox/inventory/items>. The HTTP query parameter of **filter[order]=name%20ASC** returns the list of items in ascending alphabetical order.

The consumer application sends three HTTP headers in the API request:

- The **X-IBM-Client-Id** and **X-IBM-Client-Secret** headers contain the client ID and client secret for the consumer application. The two headers satisfy the API key and secret security requirements that you defined in the inventory API definition.
- The **Authorization** header stores the OAuth access token value. You must send a valid access token for all API operations that you secured with the OAuth security requirement.

The API request flows through the architecture that you built from exercise 5 to exercise 11.

1. The consumer application calls the /inventory/items operation at apigw.think.ibm, the DataPower API Gateway.
2. The Docker container instance runs a copy of the inventory LoopBack application. It processes the GET /inventory/items operation call, and returns a list of inventory items.



## Troubleshooting

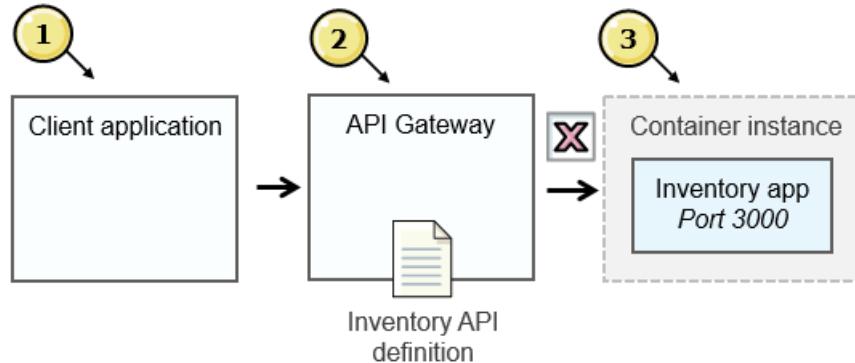
If you receive an "**Inventory API call failed**" message with a status code of **500**, the API gateway might not connect to endpoint.

Inventory API call failed:

```
{
 "name": "StatusCodeError",
 "statusCode": 500,
 "request": {
 "method": "GET",
 "url": "https://apigw.think.ibm/think/sandbox/inventory/items?filter[order]=name&ASC",
 "strictSSL": false,
 "headers": {
 "X-IBM-Client-Id": "4011e4b6775fbf1728c6977ed0ea52d3",
 "X-IBM-Client-Secret": "cd65fb95535f9b809cda8c3a68566a66",
 "Authorization": "Bearer "
 }
 },
 "response": {
 "httpCode": "500",
 "httpMessage": "URL Open error",
 "moreInformation": "Could not connect to endpoint"
 }
}
```



The failure occurs between the API gateway and the Docker container runtime instance.



You must verify the invoke policy in the inventory API definition. In addition, check that the Docker container that runs the application is started.

- Make sure that the `$(target-url)` environment property resolves to `$(app-server)${request.path}${request.search}`.
  - Make sure that the `$(app-server)` environment property resolves to `http://platform.think.ibm:3000`.
  - Make sure that the Docker container is started.

3. Ensure that the Docker container is started.

- a. Open a terminal window.
  - b. Confirm that the Docker container is started.

```
$ docker ps
```

| CONTAINER ID  | IMAGE                  | COMMAND         | CREATED        |
|---------------|------------------------|-----------------|----------------|
| STATUS        | PORTS                  | NAMES           |                |
| 8e228c99c6da  | apic-inventory-image   | "npm start"     | 14 seconds ago |
| Up 13 seconds | 0.0.0.0:3000->3000/tcp | friendly_swartz |                |

- c. If it is not started, start the Docker container.

```
$ docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000 apic-inventory-image
```

```
> inventory@1.0.0 start /inventory
```

> node .

Web server listening at: <http://localhost:3000>



## **Important**

If you *made any corrections* to the inventory API definition, you must publish your updates to the API Manager.

## End of exercise



## Exercise review and wrap-up

In the first part of the exercise, you verified the **OAuth 2.0 authorization** flow. You reviewed the settings for the **/oauth20/authorize** and **/oauth20/token** API operations.

In the second part of the exercise, you verified the **/inventory/items** API operation by calling the **inventory** API that was running in a Docker container.





Global Knowledge®

# Exercise 13. Assemble the shipping message processing policy - optional

## Estimated time

00:45

## Overview

This exercise explains how to create a message processing policy by mapping data from multiple APIs into a single response. You complete the shipping API that was imported in an earlier exercise.

## Objectives

After completing this exercise, you should be able to:

- Review an existing API definition in the assembly editor
- Map data from multiple API calls into an aggregate response
- Publish the product that contains the API
- Test the API in the Developer Portal
- Test the API with the consumer web application

## Introduction

In the previous developer portal exercise, you tested the logistics, financing, and inventory APIs with the consumer app, a client-side JavaScript application with a web interface. The shipping portion of the logistics API was purposely left out of the message processing policy exercise to reduce its duration.

In this exercise, you complete the API assembly definition for the shipping portion of the **logistics** API. In the shipping sequence of the logistics API, you define a sequence of processing policies that aggregates data from multiple sources.

## Requirements

Before you start this exercise, you must complete the data sources, remote, policies, authorization, containers, publish, and developer portal exercises in this course.

You must complete this lab exercise in the student development workstation: the Ubuntu host environment. This virtual machine is preconfigured with the Node runtime environment, the “npm” Node package manager, and the IBM API Connect toolkit.



Global Knowledge®

## Exercise instructions.

### Preface

Follow the instructions that are provided under the heading "Before you begin" in the Exercises description at the start of this guide.



## 13.1. Review and change the logistics API definition

In this section, you review the logistics API definition.

The **logistics** API provides two REST APIs:

- The **GET /stores** operation returns the address of a store location based on the postal code (ZIP code) that you provide.
- The **GET /shipping** operation returns two shipping rate quotations. The service combines, or aggregates, the responses from two REST API calls. This operation is defined but not yet completed.



**Important**

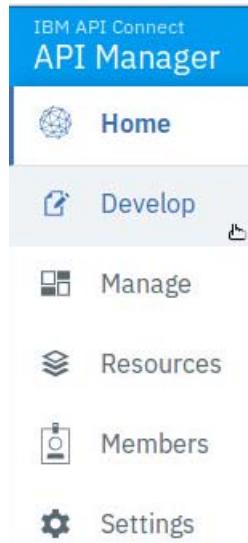
Before doing this exercise, ensure that the logistics GET /shipping API operation returns a 200 OK response message by testing the operation with the consumer web application or on the Developer Portal.

The screenshot shows the IBM API Connect Developer Portal interface. At the top, there's a navigation bar with links for 'API Products', 'Apps', 'Blogs', 'Forums', a search icon, and 'Organization Publish'. Below the navigation bar, there's a 'Support' link. The main area features a logo for 'logistics 1.0.0' with a 5-star rating. On the left, there's a sidebar with 'Overview' and 'Definitions' sections, and a 'GET /shipping' operation highlighted. The 'Request' section shows a GET request to 'https://apigw.think.ibm/think/sandbox/logistics/shipping?zip=94126' with specific headers: 'Accept: application/json', 'X-IBM-Client-Id: 06d0b874bffc2ff32b105079', and 'ddd2b5f7'. The 'Response' section shows a 200 OK status with headers: 'x-global-transaction-id: 196c55655ca24fc100000403', 'x-ratelimit-limit: name=default,100;', and 'x-ratelimit-remaining: name=default,99;'. A large watermark for 'Global Knowledge' is visible across the center of the page.

Continue with the exercise after you verify that the logistics GET /shipping operation works.

- \_\_\_ 1. Open the API Manager web application.
  - \_\_\_ a. Open a browser window.  
Then, type `https://manager.think.ibm`.
  - \_\_\_ b. Sign in to API Manager. Type the credentials:
    - User name: ThinkOwner
    - Password: Passw0rd!
 Click **Sign in**.  
You are signed in to API Manager.

- \_\_ 2. Click the tile Develop APIs and Products, or select the Develop option from the side menu.



- \_\_ 3. Review the logistics API paths and operations.
- \_\_ a. Click **logistics-1.0.0** in the list of APIs and products.

| APIs and Products                                                                                     |  | Add ▾      |
|-------------------------------------------------------------------------------------------------------|--|------------|
| TITLE                                                                                                 |  | TYPE       |
|  financing-1.0.0    |  | API (REST) |
|  inventory-1.0.0   |  | API (REST) |
|  ItemService-1.0.0 |  | API (WSDL) |
|  logistics-1.0.0   |  | API (REST) |

- \_\_ b. With the logistics API open in the editor, click the **Paths** option. The `/shipping` and `/stores` paths are displayed.
- \_\_ c. Click the `/shipping` path to open it with the editor.
- \_\_ d. The GET operation is already defined. Click the **GET** operation.
- \_\_ e. An operation ID that is named `shipping.calc` is defined. Scroll to the parameters area on the page. A parameter that is named `zip` is a required parameter.
- \_\_ f. Click the return arrow twice on the page to return to the Design view for the logistics API.

- \_\_\_ 4. Change the logistics API properties.

- \_\_\_ a. Click **Properties** in the list of options for the logistics API.

The screenshot shows the 'Properties' tab selected in the 'logistics 1.0.0' API setup. The table lists two properties:

| PROPERTY NAME           | ENCODED | DESCRIPTION                                 |
|-------------------------|---------|---------------------------------------------|
| <b>shipping_svc_url</b> | false   | Location of the shipping calculator service |
| <b>target-url</b>       | false   | The URL of the target service               |

- \_\_\_ b. In the list of properties, click **shipping\_svc\_url**.
- \_\_\_ c. The default value for the property is set to `http://shipping.think.ibm:5000/calculate`. Change the default value for the property to  
`http://platform.think.ibm:5500/calculate`

The 'Edit Property' dialog shows the following fields:

- Name:** shipping\_svc\_url
- Default value (optional):** http://platform.think.ibm:5500/calculate
- Description (optional):** Location of the shipping calculator service

- \_\_\_ d. The URL is the endpoint for the back-end service that is called by the shipping operation.
- \_\_\_ e. **Save** the change.

## 13.2. Define message processing policy for the logistics API shipping operation

In this section, define a sequence of message processing policies that calls an external web service to calculate the shipping costs.

- 1. Open the **Assemble** view.
  - a. In the API Manager, switch to the **Assemble** view for the **logistics** API definition.
  - b. An assembly is already created with a switch that contains a sequence for the shipping operation and a sequence for the stores operation.



- 2. Add an invoke policy to calculate the shipping cost with the xyz shipping company.
  - a. Add an **invoke** policy in the **case 0** branch of the switch policy.



- \_\_\_ b. In the properties view for the **invoke** policy, type the following details:
- Title: **invoke\_xyz**
  - URL: **`$(shipping_svc_url)?company=xyz&from_zip=90210&to_zip={zip}`**

The screenshot shows the 'invoke\_xyz' properties view. The 'Title' field is set to 'invoke\_xyz'. The 'URL \*' field contains the expression '\$(shipping\_svc\_url)?company=xyz&from\_zip=90210&to\_zip={zip}'. A note below the URL field states: 'The URL to be invoked.'

- Scroll down to the end of the Properties view.
- Clear **Stop on error**.
- Response object variable: **xyz\_response**

The screenshot shows the 'Stop on error' and 'Response object variable' settings. The 'Stop on error' checkbox is unchecked. The 'Response object variable' field is set to 'xyz\_response'.

- \_\_\_ c. Save and close the Invoke properties view.



### Note

The `$(shipping_svc_url)` is an environment-specific parameter that stores the network address for the shipping cost calculator service. The message processing policy calls the remote service, with the `zip` query parameter from the original GET /shipping API operation request.

- \_\_\_ 3. Add a second **invoke** policy to calculate the shipping cost with the cek shipping company.
- \_\_\_ a. In the **case 0** path, add an **invoke** policy to the *right* of the `invoke_xyz` policy.

\_\_\_ b. Type the following properties in the **invoke** policy:

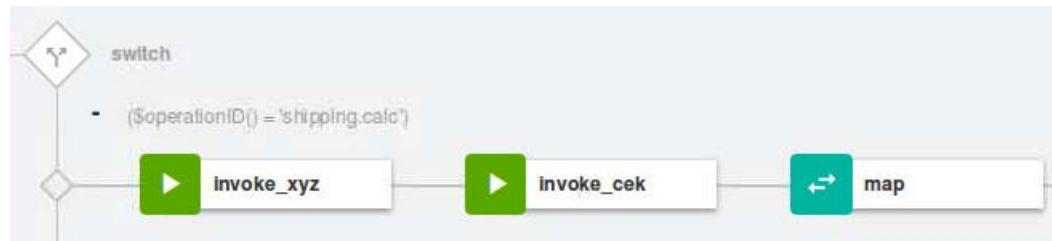
- Title: **invoke\_cek**
- URL: **`$(shipping_svc_url)?company=cek&from_zip=90210&to_zip={zip}`**

The screenshot shows the 'invoke\_cek' properties view. It includes fields for Title (set to 'invoke\_cek'), URL (set to '\$(shipping\_svc\_url)?company=cek&from\_zip=90210&to\_zip={zip}') with a note below it stating 'The URL to be invoked.'.

- Clear **Stop on error**.
- Response object variable: **cek\_response**

The screenshot shows the 'invoke\_cek' properties view again. The 'Stop on error' checkbox is unchecked. The 'cek\_response' response object variable is highlighted with a red border.

- \_\_\_ c. Save and close the Invoke properties view.
- \_\_\_ 4. Combine the results from the **invoke\_xyz** and **invoke\_cek** service calls into one response message.
- \_\_\_ a. Add the **map** policy after the second invoke policy in the **case 0** branch.



- \_\_\_ b. Open the **map** policy properties view.
- \_\_\_ c. Click the option to maximize the properties view.
- \_\_\_ d. Click the **edit** icon in the **input** column.
- \_\_\_ e. Click **+input** to add an input source.

\_\_\_ f. Specify the following input message details:

- Context variable: **xyz\_response.body**
- Name: **xyz**
- Content type: **application/json**
- Definition: **Inline schema**

\_\_\_ g. Open `lab_files/policies/schema_shippingSvc.yaml` with the **gedit** text editor.

\_\_\_ h. Copy the contents of `schema_shippingSvc.yaml`.

\_\_\_ i. Paste the clipboard contents into the **schema as YAML** window.

```

1 $schema: 'http://json-schema.org/draft-04/schema#'
2 id: 'http://jsonschema.net'
3 type: object
4 properties:
5 company:
6 id: 'http://jsonschema.net/company'
7 type: string
8 name: company
9 rates:
10 id: 'http://jsonschema.net/rates'
11 type: object
12 properties:
13 next_day:
14 id: 'http://jsonschema.net/rates/next_day'
15 type: string
16 name: next_day
17 two_day:
18 id: 'http://jsonschema.net/rates/two_day'
19 type: string
20 name: two_day
21 ground:
22 id: 'http://jsonschema.net/rates/ground'
23 type: string
24 name: ground

```

\_\_\_ j. Click **Done**.

- \_\_\_ k. Add a second **Input** message in the **map** policy with the following properties:
- Context variable: `cek_response.body`
  - Name: `cek`
  - Content type: `application/json`
  - Definition: `inline schema`

| Context variable               | Name             |
|--------------------------------|------------------|
| <code>xyz_response.body</code> | <code>xyz</code> |

| Content type                  | Definition *               |
|-------------------------------|----------------------------|
| <code>application/json</code> | <code>Inline schema</code> |

| Context variable               | Name             |
|--------------------------------|------------------|
| <code>cek_response.body</code> | <code>cek</code> |

| Content type                  | Definition *               |
|-------------------------------|----------------------------|
| <code>application/json</code> | <code>Inline schema</code> |

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

- \_\_\_ l. Copy the contents of the same `lab_files/policies/schema_shippingSvc.yaml` file into the **inline schema** window.
- \_\_\_ m. Click **Done**.
- \_\_\_ n. Click **Done**.
- \_\_\_ 5. Define an output message for the GET /shipping API operation in the **Output** column of the map policy.
- \_\_\_ a. Click the **edit (pencil)** icon in the **Output** column.
  - \_\_\_ b. Click **+output** to add an output message.

- \_\_\_ c. Type the following details in the output message:

- Context variable: `message.body`
- Name: `output`
- Content type: `application/json`
- Definition: `#/definitions/shipping`

The screenshot shows the 'map' policy editor interface. At the top, there are three icons: a green square with arrows, a blue square with a gear, and a red square with an 'X'. The word 'map' is displayed next to the first icon. Below the header, there are two rows of configuration fields.

| Context variable          | Name                |
|---------------------------|---------------------|
| <code>message.body</code> | <code>output</code> |

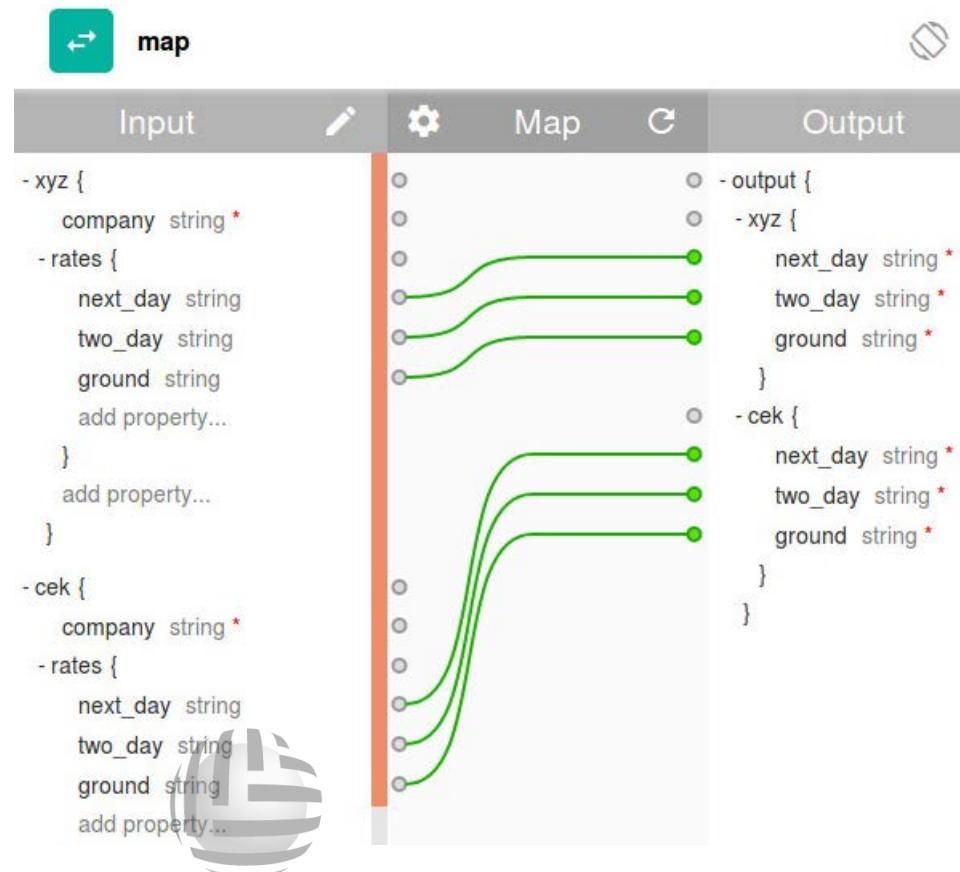
  

| Content type                  | Definition*                         |
|-------------------------------|-------------------------------------|
| <code>application/json</code> | <code>#/definitions/shipping</code> |

At the bottom left, there is a blue button labeled '+ output' and a link '+ OUTPUTS FOR OPERATION...'. On the right side, there is a blue 'Done' button and a trash can icon.

- \_\_\_ d. Click **Done**.
- \_\_\_ 6. Map the responses from the `invoke_xyz` and `invoke_cek` calls to the GET /shipping response message.
- \_\_\_ a. In the map policy editor, the `xyz_response.body` and `cek_response.body` messages appear in the `input` column. The final `output` message combines these results into one message.
  - \_\_\_ b. Draw a line from each of the fields in the `xyz` input message to the `xyz` section of the output message.

- \_\_ c. Draw a line from each of the fields in the **cek** input message to the **cek** section of the output message.



- \_\_ d. Close the **map** policy editor.  
\_\_ 7. Save the changes in the logistics API definition.

## Information

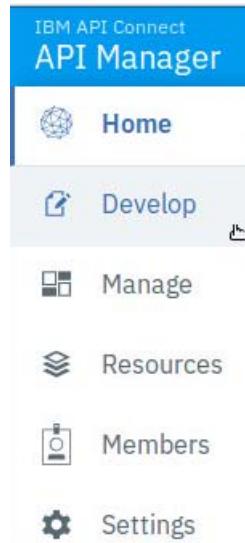
In case zero (0), the GET /shipping API operation makes two external service calls. The two **invoke** policies save a shipping cost estimate for the **xyz** and **cek** companies. The **map** policy copies the results from the two service calls into one response message.



## 13.3. Publish the API and start the back-end application

In this section, you publish the API policy assembly in the sandbox environment. You start the back-end application that is called by the `GET /shipping` operation.

- 1. Click the Develop option from the navigation menu.



- 2. Publish the product that includes the logistics API definitions.
  - a. From the list of APIs and products page, click the options ellipsis in the **think-product-1.0.0** row. Then, select **Publish**.
  - b. Select the **Sandbox** catalog publish target.

The image shows a 'Publish To' dialog box. At the top, it says 'Publish To'. Below that, under 'Catalog', 'Sandbox' is selected in a dropdown menu. There is also an unchecked checkbox for 'Publish to specific gateway services' with a note explaining that publishing to all gateway services is the default. At the bottom right are 'Cancel' and 'Publish' buttons.

Click **Publish**.

- c. The product is successfully published.

\_\_\_ 3. Start the shipping application from a terminal application.

\_\_\_ a. Open the Terminal

\_\_\_ b. Navigate to the `consumer_app` directory.

```
$ cd ~/lab_files/shipping
/home/localuser/lab_files/devportal/consumer_app/
```

\_\_\_ c. Start the shipping application.

```
$ npm start
> shipping@1.0.0 start /home/localuser/lab_files/shipping
> node ./bin/www
```

\_\_\_ d. The shipping application is started.

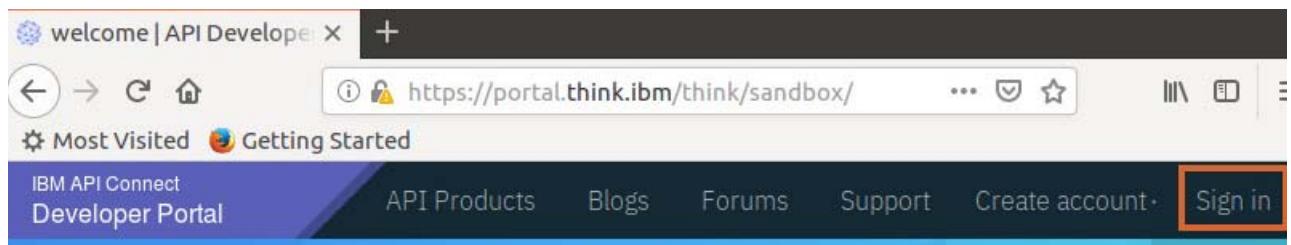
Leave the terminal running.



## 13.4. Test the logistics API policy assembly on the Developer Portal

In this section, you test the logistics GET /shipping operation on the Developer Portal.

- \_\_\_ 1. Open the Developer Portal page.
  - \_\_\_ a. In a web browser address, type:  
`https://portal.think.ibm/think/sandbox`
  - \_\_\_ b. The Developer Portal public page is displayed.
- \_\_\_ 2. Sign in to the Developer Portal with the account of the owner of the **Publish** consumer organization.
  - \_\_\_ a. Click the **Sign-in** link.



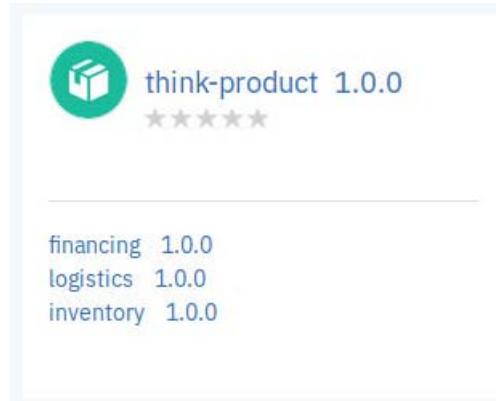
The screenshot shows a web browser window with the following details:

- Address Bar:** https://portal.think.ibm/think/sandbox/
- Toolbar:** Includes back, forward, search, and refresh buttons.
- Header:** "welcome | API Develop" and a plus sign for new tabs.
- Navigation:** "Most Visited" and "Getting Started".
- Menu:** IBM API Connect Developer Portal, API Products, Blogs, Forums, Support, Create account, and a "Sign in" button which is highlighted with a red box.
- Content:** A "Global Knowledge" watermark and a "Sign in" button.
- Form Fields:** "Username" field containing "OwnerPublish" and "Password" field containing "Passw0rd!".
- Buttons:** "Sign in" button at the bottom of the form.

Click **Sign in**.

- \_\_\_ c. The user is signed in to the Publish organization on the Developer Portal.

- \_\_\_ 3. Open the product that contains the logistics API.
  - \_\_\_ a. Click the **API Products** from the menu.
  - \_\_\_ b. Navigate to the think-product.



- \_\_\_ c. Click logistics in the list of APIs within the think-product icon.
- \_\_\_ 4. Test the logistics API shipping operation from the Developer Portal.
  - \_\_\_ a. Select the GET /shipping operation in the logistics API.



- \_\_\_ b. In the test client, click **Try it** for the GET <https://apigw.think.ibm/sales/sb/logistics/shipping> operation.
- \_\_\_ c. The client ID from the Think Application is prefilled.

- \_\_ d. Type **98121** as the **zip** input parameter.

Client ID  
Think Application

Header

Accept  
application/json

Content-Type  
application/json

Query

zip\*  
98121

Generate

Reset Send



Click **Send**.

Global Knowledge ®

- \_\_\_ e. Confirm that the GET /shipping operation combined the results from the xyz and cek shipping companies into a single result in the response message.

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Request  | <pre>GET https://apigw.think.ibm/think/sandbox /logistics/shipping?zip=98121 Headers: Accept: application/json X-IBM-Client-Id: 06d0b874bffc2ff32b105079 ddd2b5f7</pre>                                                                                                                                                                                                                                                                                                                      |
| Response | <pre>Code: 200 OK Headers: content-type: application/json x-global-transaction-id: 196c55655ca27cbe 00000232 x-ratelimit-limit: name=default,100; x-ratelimit-remaining: name=default,99; {   "xyz": {     "next_day": "39.36",     "two_day": "26.93",     "ground": "20.72"   },   "cek": {     "next_day": "35.78",     "two_day": "24.48",     "ground": "18.83"   } }</pre>  <p>Global Knowledge®</p> |



### Note

If you do not get a successful response message, you might need to resubscribe the application to the plan. Then, rerun the test in the Developer Portal.

## 13.5. Test the logistics GET /shipping API with the consumer client web application

In this section, you test all the APIs in the think-product from the consumer client web application. The consumer client web application runs an integrated test of all the APIs in the exercise case study.

- 1. Start the Loopback application in the Docker runtime environment.

- a. Open the Terminal with **Terminal > New Terminal** from the Terminal menu.
  - b. Start the Docker image that runs the inventory API.

```
$ docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
apic-inventory-image
```

```
> inventory@1.0.0 start /inventory
> node .
```

```
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- 2. Start the consumer application from another terminal application.

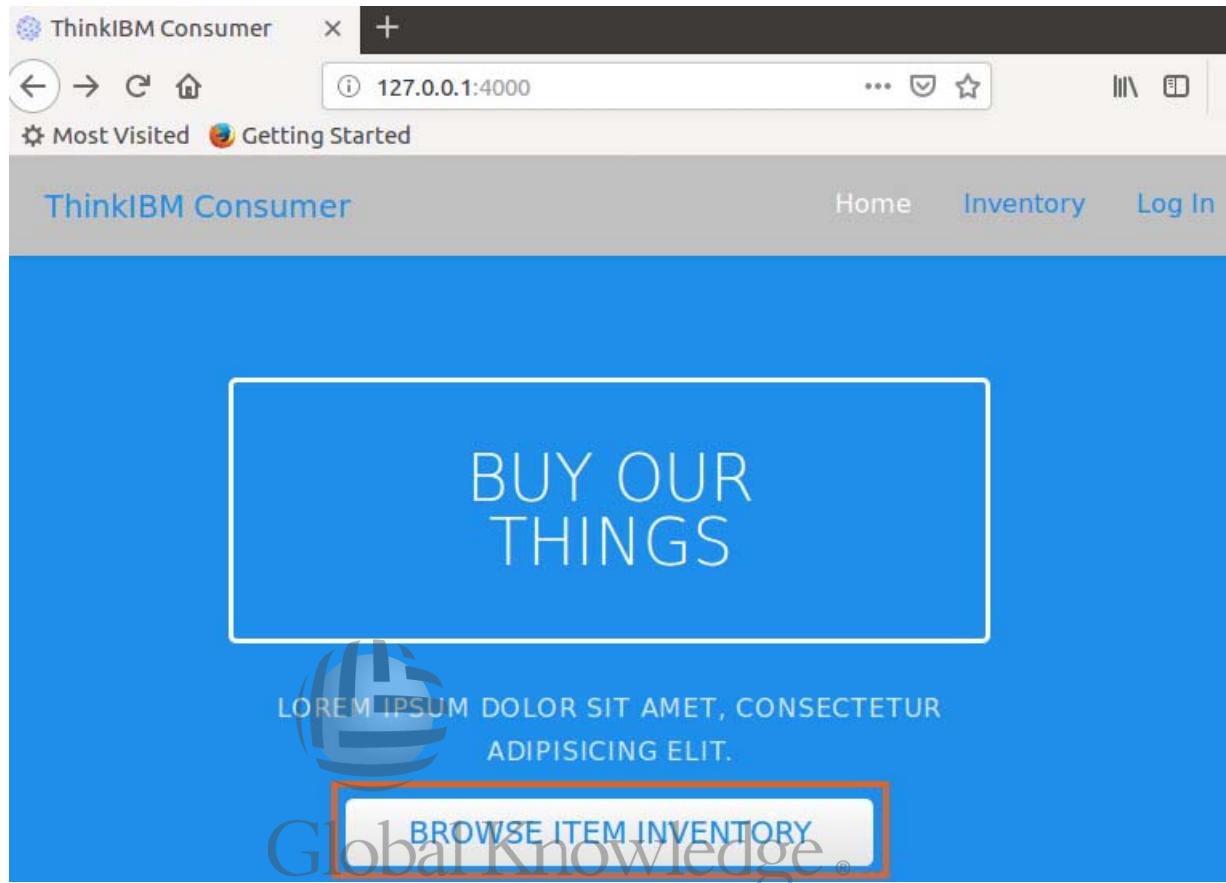
- a. Open the Terminal with **Terminal > New Terminal** from the Terminal menu.
  - b. Navigate to the `consumer_app` directory.

```
$ cd ~/lab_files/devportal/consumer_app/
$ pwd
/home/localuser/lab_files/devportal/consumer_app/
```

- c. Start the consumer application.

```
$ npm start
```

- \_\_\_ 3. Test the `GET /inventory/items` API operation from the consumer application.
- \_\_\_ a. When you start the consumer application, a web browser opens to the main page. If the consumer application does not start automatically, type `127.0.0.1:4000` in the address of a browser window.



Click **BROWSE ITEM INVENTORY**, or click **Log In**.

- \_\_\_ b. Type:
- User name: `user`
  - Password: `pass`

Click **Log In**.

**Note**

The oauth service accepts any non-blank values for the user name and password values.

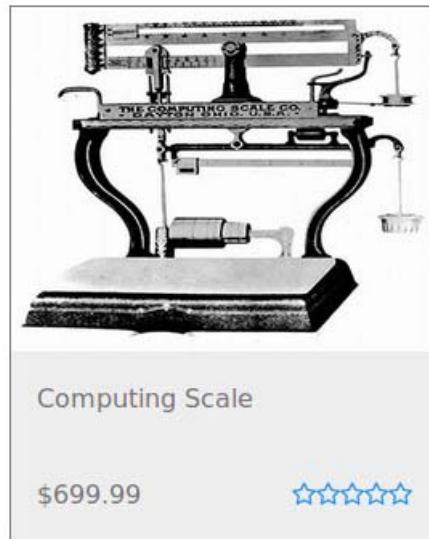
- \_\_\_ c. The page displays the list of inventory items that are retrieved from the mySQL database.

The screenshot shows a web browser window titled "ThinkIBM Consumer". The address bar indicates the URL is [127.0.0.1:4000/inventory](http://127.0.0.1:4000/inventory). The main content area displays three inventory items:

| Item            | Description                                                                                                            | Price     | Rating  |
|-----------------|------------------------------------------------------------------------------------------------------------------------|-----------|---------|
| Calculator      | A black and white photograph of a vintage-style electronic calculator with a numeric keypad and various function keys. | \$5199.99 | 5 stars |
| Collator        | A black and white photograph of a large industrial collator machine, likely used for sorting and stacking documents.   | \$2799.99 | 5 stars |
| Computing Scale | A black and white illustration of a mechanical computing scale with a balance beam and weights.                        | \$699.99  | 5 stars |

- \_\_\_ 4. Confirm that the client application displays inventory items in the page. Next, you test the other features of the case study from the consumer client application.

- \_\_\_ 5. Test the logistics GET /shipping function in the consumer client application.
- \_\_\_ a. Click the Computing Scale image on the web page, or any other inventory image.



- \_\_\_ b. The detail page is displayed.

A screenshot of the ThinkIBM Consumer website. At the top, there is a navigation bar with links for 'Home', 'Inventory', and 'Log in'. Below the navigation bar, there is a large product image of a 'Computing Scale'. To the right of the image, the product title 'Computing Scale' is displayed in bold, along with its price '\$699.99' and a five-star rating icon. Further down, there is a 'Calculate Shipping' button and a 'Zip Code' input field. At the bottom of the page, there is a 'Calculate Monthly Payment' link and a blue 'Calculate' button.

- \_\_\_ c. Type 98121 in the zip code area under the calculate shipping title.

- \_\_\_ d. You see the same result that you saw earlier when the GET /shipping operation was tested on the Developer Portal.



- \_\_\_ e. You have successfully completed the API Connect case study and the optional GET /shipping mapping exercise.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you completed the GET /shipping subflow in the logic construct of an assembly flow.

In the GET /shipping operation of the logistics API, you combined the shipping rates from two separate remote services calls into one single REST API response. You tested the GET /shipping operation against a back-end service on the Developer Portal. Finally, you tested the operation in the integrated consumer web application that called all the APIs that you created in the case study exercises.



# Appendix A. Solutions

The **lab files** directory (`/home/localuser/lab_files`) contains source code and configuration files for the lab exercises. Each subdirectory in the lab files directory represents an exercise in the course. The model solutions for the exercises are not numbered, but rather share a common name with the exercise title, in most cases. Some exercises have no model solution. In these cases, some YAML source files are captured. Students are required to do some configuration in either the API Manager or Developer Portal user interface.

The first four exercises help to familiarize students with the course environment and user interfaces. The course case study starts with exercise 5. All exercises from exercise 5 onwards must either be completed or the solutions must be loaded to ensure a successful completion of the case study in exercise 11.

Within each exercise subdirectory, the **complete** folder contains a copy of the model solution for the lab. For example, the solution for Exercise 5, "Defining LoopBack data sources" is located in `/home/localuser/lab_files/datasources/complete`.

If you are unable to complete a particular lab exercise, back up your current work. Copy the model solution application project into your home directory. For complete instructions on how to set up a model solution for a lab exercise, refer to the `README.md` file in the corresponding **complete** directory.

*Table 9. Model solution file for course exercises*

| Exercise name                                                                                  | Completed exercise solution                     |
|------------------------------------------------------------------------------------------------|-------------------------------------------------|
| <a href="#">Exercise 1, "Review the API Connect development and runtime environment"</a>       | <i>Not applicable</i>                           |
| <a href="#">Exercise 2, "Create an API definition from an existing API"</a>                    | <code>~/lab_files/interface/complete</code>     |
| <a href="#">Exercise 3, "Define an API that calls an existing SOAP service"</a>                | <code>~/lab_files/existing/complete</code>      |
| <a href="#">Exercise 4, "Create a LoopBack application"</a>                                    | <code>~/lab_files/loopback/complete</code>      |
| <a href="#">Exercise 5, "Define LoopBack data sources"</a>                                     | <code>~/lab_files/datasources/complete</code>   |
| <a href="#">Exercise 6, "Implement event-driven functions with remote and operation hooks"</a> | <code>~/lab_files/remote/complete</code>        |
| <a href="#">Exercise 7, "Assemble message processing policies"</a>                             | <code>~/lab_files/policies/complete</code>      |
| <a href="#">Exercise 8, "Declare an OAuth 2.0 provider and security requirement"</a>           | <code>~/lab_files/authorization/complete</code> |
| <a href="#">Exercise 9, "Deploy an API implementation to a container runtime environment"</a>  | <i>Not applicable</i>                           |
| <a href="#">Exercise 10, "Define and publish an API product"</a>                               | <i>Not applicable</i>                           |

Table 9. Model solution file for course exercises

| Exercise name                                                            | Completed exercise solution |
|--------------------------------------------------------------------------|-----------------------------|
| <a href="#">Exercise 11, "Subscribe and test APIs"</a>                   | <i>Not applicable</i>       |
| <a href="#">Exercise 12, "Troubleshooting the case study - optional"</a> | <i>Not applicable</i>       |



# Appendix B. General troubleshooting issues when working on the exercises

This part describes some issues that you might encounter when working through the exercises.

## Timeout in the user interface

If you leave the user signed-on to the user interface, for example, the API Manager, for a significant amount of time, such as overnight, then when you try to continue working in the user interface, you see this message:



**Solution:** Close the browser and sign on again.

## IBM Remote Lab Platform-specific issues - suspension of images

The IRLP Skytap environment suspends the virtual machines after 1 hour of inactivity. This can lead to the DataPower and the IBM Connect system on the Ubuntu VM getting out of synch.

If you get an unauthorized error with the body content:

```
{
 "httpCode": "401",
 "httpMessage": "Unauthorized",
 "moreInformation": "The API request is rejected because of no API Connect
Gateway Service."
}
```

This error might result from the 2 VMs getting out of synch.

**Solution:** Restart the DataPower gateway.

Students must **shutdown their VMs at the end of each day** and restart them the next day to circumvent this problem.

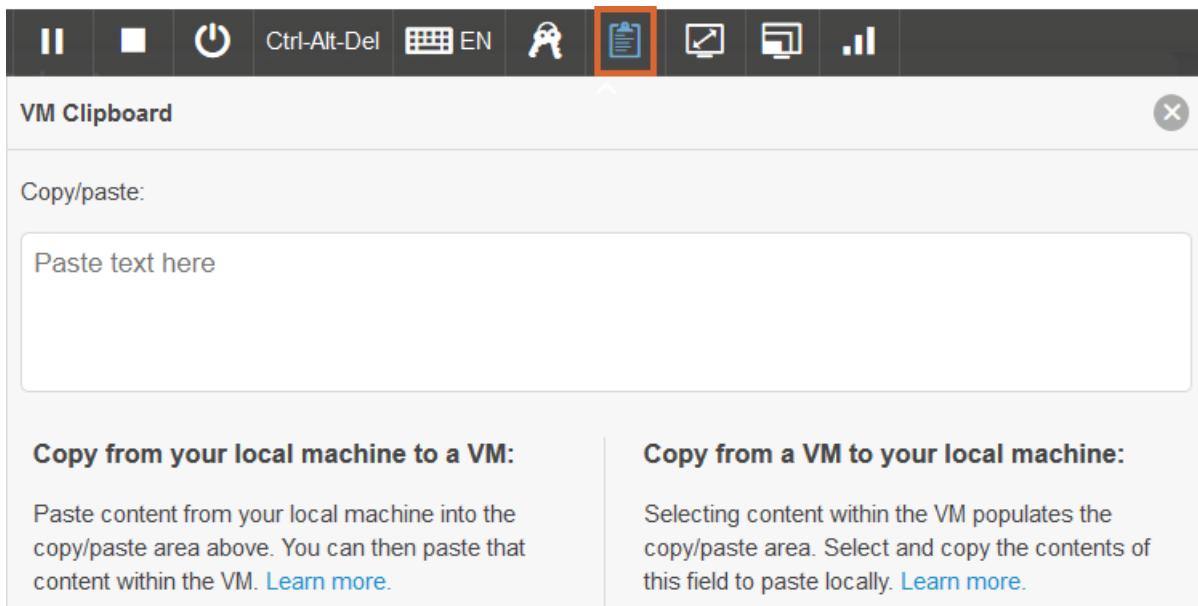


### Note

When the student image is started from the suspended state it might take about 4 minutes before the sign-on page accepts the password.

## IBM Remote Lab Platform-specific issues - cut and paste

The IRLP Skytap environment uses the Clipboard as go-between for the cut and paste operations.



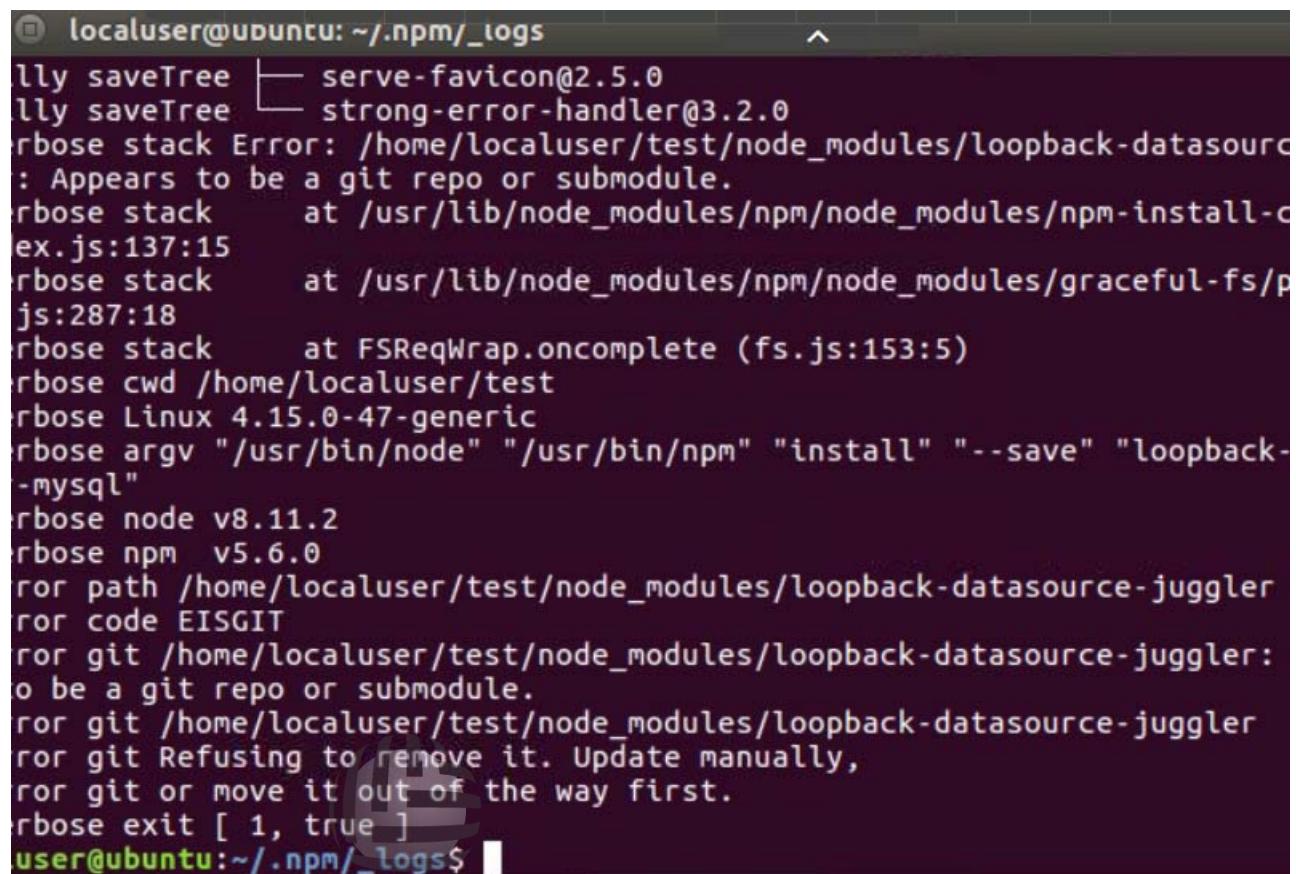
Copy the code that you want to the clipboard and verify that it is indeed visible in the Clipboard. Then, from the Clipboard select Copy.

Paste the contents into the target.

## Issues when installing the LoopBack Connectors

Since the LoopBack connectors that are used in the class are downloaded from the Internet, if a new version of the connector is uploaded while the class is in progress, you might get a situation where the installation of the LoopBack connector fails.

For example, if the command `npm install loopback-connector-mysql` fails, you see an error in the terminal.



```

localuser@ubuntu: ~/npm/_logs
 lly saveTree └─ serve-favicon@2.5.0
 lly saveTree └─ strong-error-handler@3.2.0
 rbose stack Error: /home/localuser/test/node_modules/loopback-datasource-juggler: Appears to be a git repo or submodule.
 rbose stack at /usr/lib/node_modules/npm/node_modules/npm-install-cmd/ex.js:137:15
 rbose stack at /usr/lib/node_modules/npm/node_modules/graceful-fs/promises.js:287:18
 rbose stack at FSReqWrap.oncomplete (fs.js:153:5)
 rbose cwd /home/localuser/test
 rbose Linux 4.15.0-47-generic
 rbose argv "/usr/bin/node" "/usr/bin/npm" "install" "--save" "loopback-connector-mysql"
 rbose node v8.11.2
 rbose npm v5.6.0
 ror path /home/localuser/test/node_modules/loopback-datasource-juggler
 ror code EISGIT
 ror git /home/localuser/test/node_modules/loopback-datasource-juggler: Appears to be a git repo or submodule.
 ror git /home/localuser/test/node_modules/loopback-datasource-juggler
 ror git Refusing to remove it. Update manually,
 ror git or move it out of the way first.
 rbose exit [1, true]
user@ubuntu:~/npm/_logs$

```

**Solution:** Remove the older connector and reinstall.

```

cd node_modules
rm -rf loopback-datasource-juggler

```

Then, move back to the application directory and install the appropriate connector. For example:

```
npm install loopback-connector-mysql
```

## Issues when building the Docker image

Some Docker components are downloaded when you build the Docker image. If you are unable to build the Docker image according to the exercise instructions, review the information shown here:



### Information

You might receive an error when running the `build` command, such as

```

fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/community/x86_64/APKINDEX.tar.gz
ERROR: http://dl-cdn.alpinelinux.org/alpine/v3.8/community: temporary error (try again later)

```

If you get the fetch error, run these steps:

Find your DNS IP address with the command:

```
$ nmcli dev show | grep 'IP4.DNS'
```

```
IP4.DNS[1]: 9.24.218.75
```

Create the file /etc/docker/daemon.json and include the DNS IP address in the list:

```
$ sudo nano /etc/docker/daemon.json
$ [sudo] password for localuser: passw0rd
{
 "dns": ["8.8.8.8", "8.8.4.4", "10.0.0.10", "9.24.218.75"]
}
```

Write out the file.

Restart the Docker daemon with the command:

```
sudo service docker restart
```

Rerun the docker build command.

If the build command still fails, you can download the image from Docker Hub with the command:

```
docker pull kevinom/apic-inventory-image:2018.4.1.1
```

Then, replace apic-inventory-image with

kevinom/apic-inventory-image:2018.4.1.1 in the remainder of the exercises.

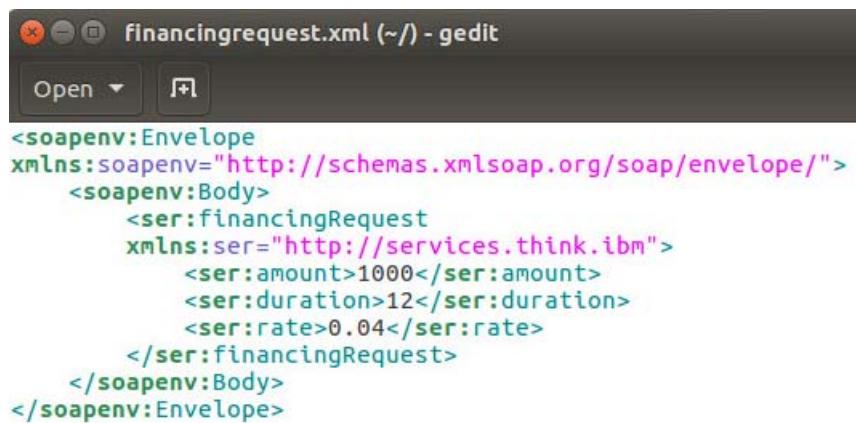
## Testing that the financing back-end service is working on DataPower.

The financing application runs as a service in the Services domain on the DataPower VM.

You can test that the service is running on DataPower with the following curl command:

```
curl -v -k -H "Content-Type:application/xml" POST
https://apigw.think.ibm:1443/financing --data-binary "@financingrequest.xml"
```

The financing request file is located in the ~/lab\_files/policies folder, and is displayed here:



```
financingrequest.xml (~/) - gedit
Open +
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
 <ser:financingRequest
 xmlns:ser="http://services.think.ibm">
 <ser:amount>1000</ser:amount>
 <ser:duration>12</ser:duration>
 <ser:rate>0.04</ser:rate>
 </ser:financingRequest>
 </soapenv:Body>
</soapenv:Envelope>
```

Students with a knowledge of DataPower can sign on to the DataPower graphical interface and set the log option to debug and the probe option on for the Services MPGW. These students can then see the request and response messages in DataPower Probe when they call the financing application.

- \_\_\_ 1. Sign on to the DataPower gateway.
  - \_\_\_ a. In a browser, type `https://apigw.think.ibm:9090/`
  - \_\_\_ b. Then, type:
    - User name: `admin`
    - Password: `passw0rd`
    - Domain: `Services`
- \_\_\_ Click **Login**.
- \_\_\_ 2. Set debug on,
  - \_\_\_ a. In the Services domain, select the **Administration** option. Then, select **Debug**. Then, click **Troubleshooting**.
  - \_\_\_ b. Set the log level to **debug**.

The screenshot shows the 'Troubleshooting' section of the DataPower interface. At the top, there are tabs for Main, Probe, and Conformance Validation. Below the tabs, there's a 'Networking' section with a 'Ping Remote' button and a 'Remote Host' input field. Underneath is a 'Use IP version' dropdown set to 'default'. The main area is titled 'Global Knowledge' with a registered trademark symbol. Below this, there's a 'Logging' section with a 'Set Log Level' button. A dropdown menu is open over this button, showing options like 'debug' (which is highlighted with a red box) and 'info'. There's also a 'View System Logs' link and a 'Log Level' button.

- \_\_\_ 3. Set the probe on in the Services domain.
  - \_\_\_ a. Click **Services** from the navigation menu.
  - \_\_\_ b. Click the **Services MPGW** link.

The screenshot shows the 'All Services' page. At the top, there's a 'New Service' button and a refresh icon. Below is a table with columns: Service, Status, Service Type, and Front side URL. One row is visible, representing the 'Services MPGW' service. The 'Service' column shows 'Services MPGW', the 'Status' column shows a green circle with 'Up', the 'Service Type' column shows 'Multi-Protocol Gateway', and the 'Front side URL' column shows 'https://0.0.0.0:1443'. A red box highlights the 'Services MPGW' link in the 'Service' column. Below the table, it says 'Host stub services for APIC PoT'.

- \_\_\_ c. With the Services MPGW open on the page, click **Actions**. Then, select **Show Probe**.

The screenshot shows the 'Services MPGW Multi-Protocol Gateway' configuration page. At the top right, there is a 'Status: up' indicator and an 'Actions' button with a dropdown menu. The 'Actions' menu is open, showing options like 'Export', 'View Log', 'View Status', 'Show Probe', and 'Validate Conformance'. Below the status bar, there are tabs for 'General', 'Advanced', 'Subscriptions', and 'Policy'. The 'General Configuration' section contains fields for 'Multi-Protocol Gateway Name' (set to 'Services MPGW') and 'XML Manager' (set to 'default').

- \_\_\_ d. The Probe window is displayed. Click **Enable Probe**.

The screenshot shows the 'Transaction List for Services MPGW - Mozilla Firefox' window. It displays a table with columns: view, trans#, type, inbound-url, outbound-url, rule, and client-ip. A message at the bottom says '(no transaction recorded)'. At the top, there are buttons for Refresh, Flush, Enable Probe, Export Capture, View Log, Send Message, and Close. The 'Enable Probe' button is highlighted with a red box.

- \_\_\_ e. Call the financing API from the gateway as you did in the exercises.  
 \_\_\_ f. Click Refresh in the Probe window.  
 \_\_\_ g. The transaction is displayed in the Probe window.  
 \_\_\_ h. Click the magnifying icon in the Probe to see the details of the request and response messages.

The screenshot shows the same 'Transaction List for Services MPGW - Mozilla Firefox' window. A transaction entry is visible in the table:  
 view: 4017  
 trans#: request  
 type: https://10.0.0.20:1443/financing  
 inbound-url: https://10.0.0.20:1443/financing  
 rule: Services MPGW\_Policy\_ShippingCalc  
 client-ip: (no transaction recorded)  
 There is a magnifying glass icon next to the transaction entry.

- \_\_\_ i. Disable the Probe when you are finished.  
 \_\_\_ j. Set Debug to error.

**Important**

You can save changes to the Services domain related to the debugging level. However, do not save any changes that might be flagged on the apiconnect domain. API Manager manages these changes.

## B.1. Useful Kubernetes information and commands

This part describes some resources for learning about Kubernetes and some basic Kubernetes commands that can be used on the student image.

API Connect V2018 and later runs on the Kubernetes environment. The video describes some kubectl commands that can be run on the student image to see the available resources.

### Useful Kubernetes commands.

[https://www.youtube.com/watch?v=W5xHec3\\_Tts](https://www.youtube.com/watch?v=W5xHec3_Tts)

### Extracting the API Manager logs

**Note**

Global Knowledge ®

You can review the API Connect logs from the appropriate pod that is running in Kubernetes.

Open a terminal window. Then, type:

```
kubectl get pods -n apiconnect
```

The list of running pods is displayed. Locate the pod with apim in the name and copy the associated pod name.

| localuser@ubuntu: ~/inventory                     |      |     |           |  |
|---------------------------------------------------|------|-----|-----------|--|
|                                                   |      |     |           |  |
| 0                                                 | 4d6h |     |           |  |
| r674f0bc86d-apiconnect-cc-repair-1549328400-dbs8z |      | 0/1 | Completed |  |
| 0                                                 | 3d2h |     |           |  |
| r674f0bc86d-apiconnect-cc-repair-1549328400-llww4 |      | 0/1 | Error     |  |
| 0                                                 | 3d2h |     |           |  |
| r674f0bc86d-apiconnect-cc-repair-1549328400-n9277 |      | 0/1 | Error     |  |
| 0                                                 | 3d2h |     |           |  |
| r674f0bc86d-apiconnect-cc-repair-1549328400-pzk9t |      | 0/1 | Error     |  |
| 0                                                 | 3d2h |     |           |  |
| r674f0bc86d-apiconnect-cc-repair-1549501200-v2phs |      | 0/1 | Completed |  |
| 0                                                 | 31h  |     |           |  |
| r674f0bc86d-apiconnect-cc-repair-1549501200-zx7xw |      | 0/1 | Error     |  |
| 0                                                 | 31h  |     |           |  |
| r674f0bc86d-apim-schema-init-job-ltcpz            |      | 0/1 | Completed |  |
| 0                                                 | 9d   |     |           |  |
| r674f0bc86d-apim-v2-88674dd9f-g8r95               |      | 1/1 | Running   |  |
| 12                                                | 9d   |     |           |  |
| r674f0bc86d-client-dl-srv-b7fdf9767-6g4qz         |      | 1/1 | Running   |  |
| 11                                                | 9d   |     |           |  |

Then, from the terminal, type:

```
kubectl logs -n apiconnect [pod-name] > apim.logs
```

The log file is written to the current directory.

In the terminal, type:

```
tail -1000 apim.logs
```



The log file is displayed.

```
localuser@ubuntu: ~/inventory
6e35177] =====
=====
Sat, 09 Feb 2019 00:19:05 GMT apim:routesc:rBACHelper [28f7682efc8a7c04d2ab2cbd5
6e35177] \/\ Entering: rBACHelper::checkIfUserIsOwnerBasedOnRegistrationType
Sat, 09 Feb 2019 00:19:05 GMT apim:routesc:rBACHelper [28f7682efc8a7c04d2ab2cbd5
6e35177] - internal invoker call, exiting...
Sat, 09 Feb 2019 00:19:05 GMT apim:routesc:rBACHelper [28f7682efc8a7c04d2ab2cbd5
6e35177] \/\ Exiting: rBACHelper::checkIfUserIsOwnerBasedOnRegistrationType
Sat, 09 Feb 2019 00:19:05 GMT apim:routesc:rBACHelper [28f7682efc8a7c04d2ab2cbd5
6e35177] -----
Sat, 09 Feb 2019 00:19:05 GMT apim:routes:webhook [28f7682efc8a7c04d2ab2cbd56e35
177] /\ Exiting: webhook::getPreHook
Sat, 09 Feb 2019 00:19:05 GMT apim:routes:webhook [28f7682efc8a7c04d2ab2cbd56e35
177] -----
Sat, 09 Feb 2019 00:19:05 GMT audit [28f7682efc8a7c04d2ab2cbd56e35177] =====
=====
=====
Sat, 09 Feb 2019 00:19:05 GMT audit [28f7682efc8a7c04d2ab2cbd56e35177] Successfu
l 200 response (GET /api/catalogs/7ca9e966-0426-4521-a011-cfce7c029aad/bf54761e-
8440-4bb3-9b24-aab616bb7aa0/webhooks/950d2d29-dd3c-4bfa-ac45-4f78a182f62b)
Sat, 09 Feb 2019 00:19:05 GMT audit [28f7682efc8a7c04d2ab2cbd56e35177] =====
=====
=====
```



Global Knowledge®



IBM Training



Global Knowledge.®



© Copyright International Business Machines Corporation 2019.