

Course Exercises Guide

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

Course code WD514 / ZD514 ERC 5.0



Contents

Trademarks	v
Exercises description	11
1.1. Course exercise files	1-2
1.2. Before you begin.....	1-3
1.2.1. Working with Kubernetes certificates	1-3
1.2.2. Suspending the lab environment	1-4
1.2.3. Restarting the lab environment	1-5
1.2.4. Resetting the lab environment	1-7
1.2.5. Accessing the virtual images through Remote Desktop	1-8
1.2.6. Working with the Unity desktop	1-9
Exercise 1. Reviewing the API Connect development and runtime environment	1-1
Exercise instructions	2
1.1. Starting your lab environment	1-3
1.1. To manage the virtual images in your lab environment	1-3
1.2. To start the virtual images	1-3
1.3. To access the Ubuntu image	1-5
1.2. Review the network connectivity and domains	1-6
1.3. Review the Kubernetes certificates	1-8
1.4. Review the Kubernetes runtime environment	1-11
1.5. Review resources in Cloud Manager	1-14
Exercise 2. Creating an API definition from an existing API	2-1
2.1. Review the Petstore REST API sample application	2-3
2.2. Create an API definition	2-10
2.3. Invoke a GET operation in the existing API implementation	2-20
2.4. Test the API operation in the assembly	2-23
2.5. Define a POST API operation with a JSON request message	2-29
2.6. Test the petstore API with the assembly test feature	2-35
Exercise 3. Defining an API that calls an existing SOAP service	3-1
3.1. Review the existing SOAP web service	3-2
3.2. Create a SOAP API definition from a WSDL document	3-7
3.3. Test the API on the DataPower Gateway	3-15
Exercise 4. Create a LoopBack application	4-1
4.1. Create a LoopBack application with the apic command line utility	4-2
4.2. Examine the structure of a LoopBack application	4-3
4.3. Import the API definition into the API Manager web application	4-9
4.4. Review the API definition in the API Manager web application	4-12
4.5. Test the API operation with the LoopBack API Explorer	4-19
Exercise 5. Define LoopBack data sources	5-1
5.1. Create the inventory application	5-2
5.2. Create the MySQL data source	5-3
5.3. Create the item LoopBack model	5-5
5.4. Test the inventory application and items model	5-11
5.5. Create a MongoDB data source and the review model	5-15
5.6. Define a relationship between the item and review models	5-19

5.7. 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	6-2
6.2. Create an operation hook	6-3
6.3. Test the operation hook	6-6
6.4. Create a remote hook	6-8
6.5. Test the remote hook	6-15
Exercise 7. Assemble message processing policies	7-1
7.1. Import the inventory API definition into API Manager	7-3
7.2. Test the inventory API definition by calling it on the gateway	7-10
7.3. Create the financing API definition	7-15
7.4. Create an API operation in the financing API	7-20
7.5. Set activity logging in the financing API	7-23
7.6. Invoke a SOAP web service with a message processing policy	7-25
7.7. Test the financing APIs by calling it on the gateway	7-34
7.8. Create the logistics API definition	7-37
7.9. Edit the logistics API definition	7-39
7.10. Define the message policies for the GET /stores API operation	7-42
7.11. Test the stores API by calling them on the gateway	7-47
Exercise 8. Declare an OAuth 2.0 provider and security requirement	8-1
8.1. Create a Native OAuth Provider	8-2
8.2. Configure OAuth 2.0 authorization in the inventory API	8-12
8.3. Create a test application	8-15
8.4. Test OAuth security in the inventory API	8-17
Exercise 9. Deploy an API implementation to a container runtime environment	9-1
9.1. Test a local copy of the LoopBack API application	9-3
9.2. Set up the Docker image configuration	9-5
Exercise 10. Define and publish an API product	10-1
10.1. Create a product for all three APIs	10-3
10.2. Verify that the target URL in the inventory API routes to the Docker image	10-10
10.3. Enable API key security in the financing and logistics API	10-12
10.4. Publish the product and API definitions	10-14
Exercise 11. Subscribe and test APIs	11-1
11.1. Review the consumer organizations in the Sandbox catalog	11-3
11.2. Register a client application in the Developer Portal	11-6
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
Appendix A. Solutions	A-1
Appendix B. Troubleshooting Issues	B-1
IBM Remote Lab Platform - suspension of images	1
B.1. Useful Kubernetes information and commands	B-16

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, "Reviewing the API Connect development and runtime environment"](#)
- [Exercise 2, "Creating an API definition from an existing API"](#)
- [Exercise 3, "Defining 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"](#)
- [Appendix A, "Solutions"](#)
- [Appendix B, "Troubleshooting Issues"](#)

In the exercise instructions, you can check off the line before each step as you complete it to track your progress. Most exercises include required sections, which should always be completed. It might be necessary to complete these sections before you can start later exercises.

1.1. 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:

- For the Ubuntu operating system: /home/localuser/lab_files

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



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/solutions`). 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>

1.2. Before you begin

You must review this section before you begin the exercises. This section include instructions for:

- ["Working with Kubernetes certificates"](#)
- ["Suspending the lab environment"](#)
- ["Restarting the lab environment"](#)
- ["Resetting the lab environment"](#)
- ["Accessing the virtual images through Remote Desktop"](#)
- ["Working with the Unity desktop"](#)

1.2.1. Working with Kubernetes certificates

API Connect runs on Kubernetes. Client certificates that are generated by Kubernetes expire after 1 year by default. At the time this course was created, certificate rotation was not implemented. Due to certificate expiration, the date in the environment is set to January 25, 2020. You review the certificates in the first exercise.



Important

Do not change the date or time on the Ubuntu VM. Doing so corrupts the environment. You will need to contact Learner Support (ipsupp@us.ibm.com) to provision a new VMWare image. This process takes 1-2 days to complete.

The date and time is configured in the BIOS settings for the virtual machines. Additionally, the VMs are configured not to get the current time from the Internet. Do NOT change any of these date and time settings.

1.2.2. Suspending the lab environment



Important

Do not **Shut Down** the WD514_Ubuntu VM after starting it. Due to certificate expiration, the date in the environment has been reset to January 25, 2020. If you **Shut Down** the VM, the date is reset to January 25, 2020, the labs will not work and you will need to provision a new VMWare image from Learner Support which will take 1-2 days.

When you are finished for the day, follow these instructions to ensure proper synchronization between API Connect and the IBM DataPower Gateway appliance.

- 1. Click **Suspend this VM** on the WD514_Ubuntu image.



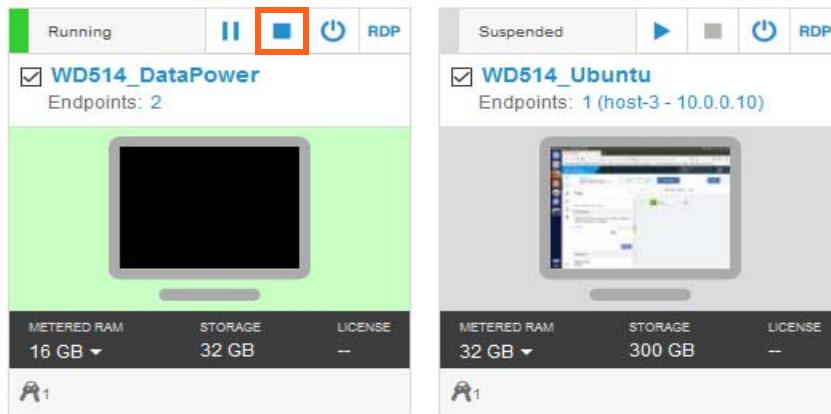
Upon VM suspension, the clock pauses at the specific date and time that you suspended the VM. The date and time remains paused while the VM is in a suspended state.



Troubleshooting

If the VM appears to be busy for several minutes without responding, try refreshing the browser to return the current state of the image.

- __ 2. After the Ubuntu virtual machine is in a suspended state, click the **Shutdown this VM** icon on the WD514_DataPower image.

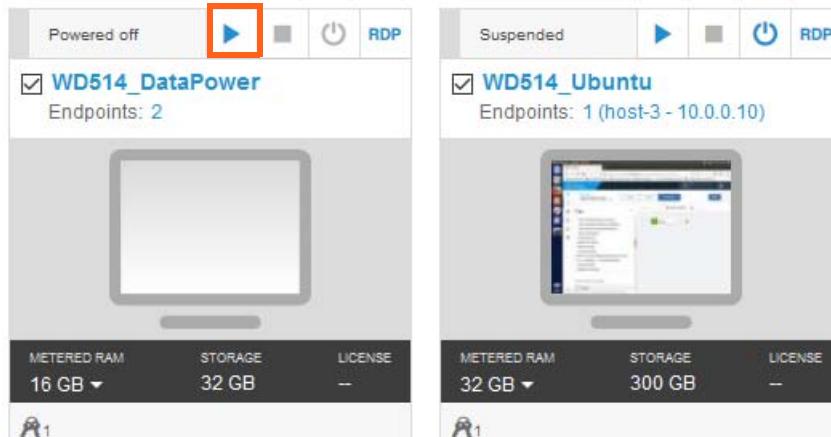


In summary, only Suspend the Ubuntu image. You can Shut Down the DataPower image. Never Shut Down the Ubuntu image.

1.2.3. Restarting the lab environment

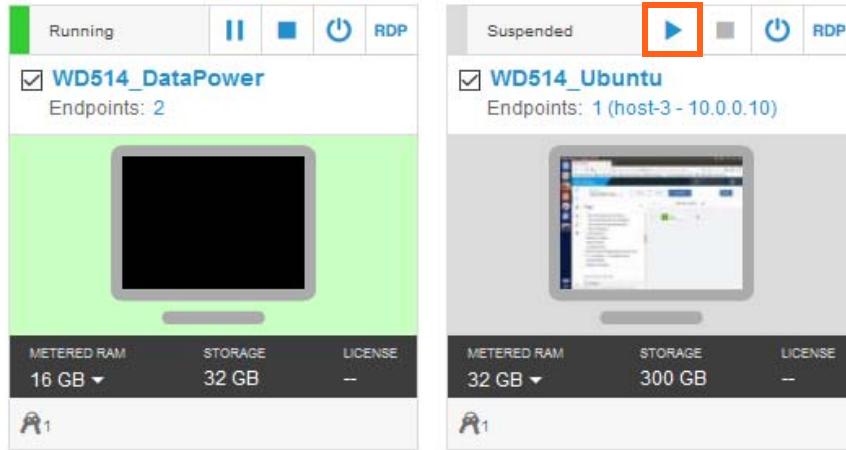
When you restart the environment, start the DataPower virtual machine first. It must be running before you start the Ubuntu virtual machine. This ensures proper synchronization.

- __ 1. Verify that the **WD514_DataPower** VM is in a Powered off state and the **WD514_Ubuntu** VM is in a suspended state.
- __ 2. Click the **Run this VM** icon on the WD514_DataPower image.

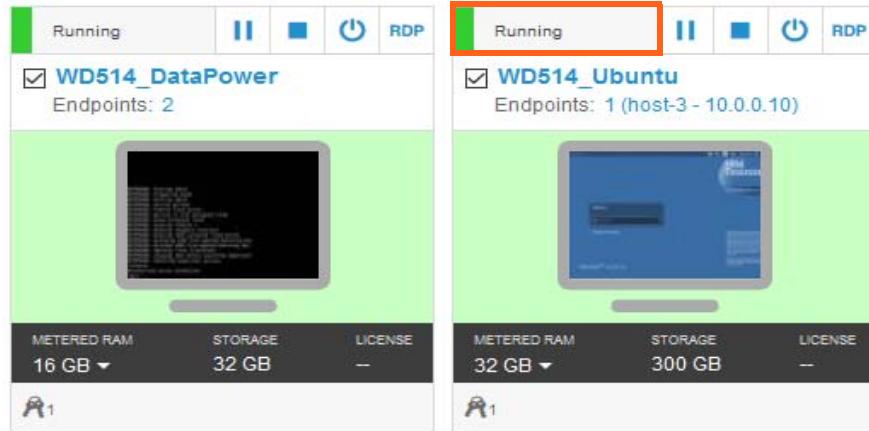


- __ 3. Wait for the DataPower image status to change to **Running**.

- ___ 4. Click the **Run this VM** icon on the WD514_Ubuntu image.



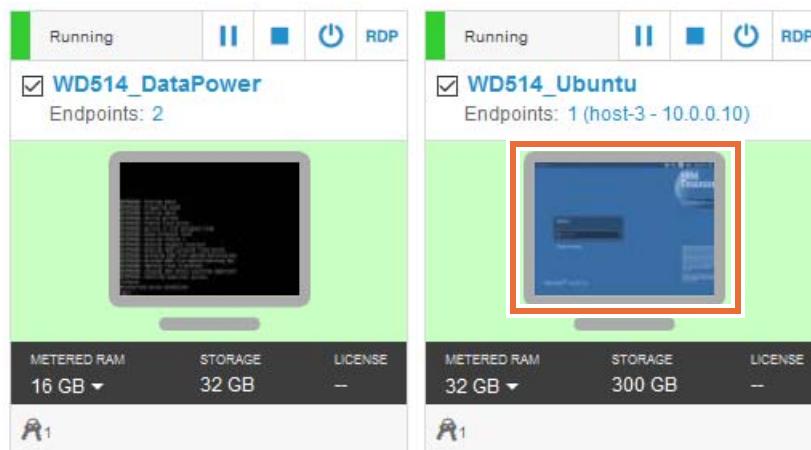
- ___ 5. Wait for the Ubuntu image status to change to **Running**.



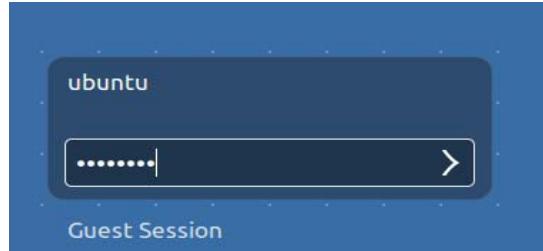
- ___ 6. Wait 10 minutes to allow the synchronization to complete and for all pods to start.

- ___ 7. Open the Ubuntu image.

- ___ a. Click the Ubuntu desktop.



- ___ b. Sign in to the Ubuntu virtual machine with the password: passw0rd



1.2.4. Resetting the lab environment

The lab environment suspends the virtual machines after 10 hours of inactivity, which causes the DataPower and the API Connect system on the Ubuntu VM to get out of sync.

When this happens, you must reset the environment.

To reset synchronization if the environment becomes suspended:

- ___ 1. Suspend and shutdown the virtual machines.
 - ___ a. Click the **Suspend this VM** icon on the WD514_Ubuntu image.
 - ___ b. Wait for the WD514_Ubuntu VM to enter a suspended state.
 - ___ c. Click **Shutdown this VM** on the WD514_DataPower image.
- ___ 2. Restart the virtual machines.
 - ___ a. Verify that WD514_DataPower is powered off and WD514_Ubuntu is in a suspended state.
 - ___ b. Click the **Run this VM** icon on the WD514_DataPower image.
 - ___ c. Wait for the DataPower image status to change to **Running**.
 - ___ d. After the DataPower image is running, click the **Run this VM** icon on the WD514_Ubuntu image.
 - ___ e. Wait for the Ubuntu image status to change to **Running**.
 - ___ f. Wait 10 minutes to allow the synchronization to complete and for all pods to start.
- ___ 3. Open the Ubuntu image.
 - ___ a. Click the Ubuntu desktop.
 - ___ b. Sign in to the Ubuntu virtual machine with the password: passw0rd

For more information, see [Appendix B, "Troubleshooting Issues"](#).



Questions

What happens when you start the VMs from a suspended state?

If you suspend the VM, the time now continues. For example, if the time on the VMs was January 25, 10:00 AM EST when you suspended the VMs, the clock stays at 10:00 AM EST for as long as the VMs are suspended. So, for example, if the VMs are suspended for three hours, the time is still January 25, 10:00 AM

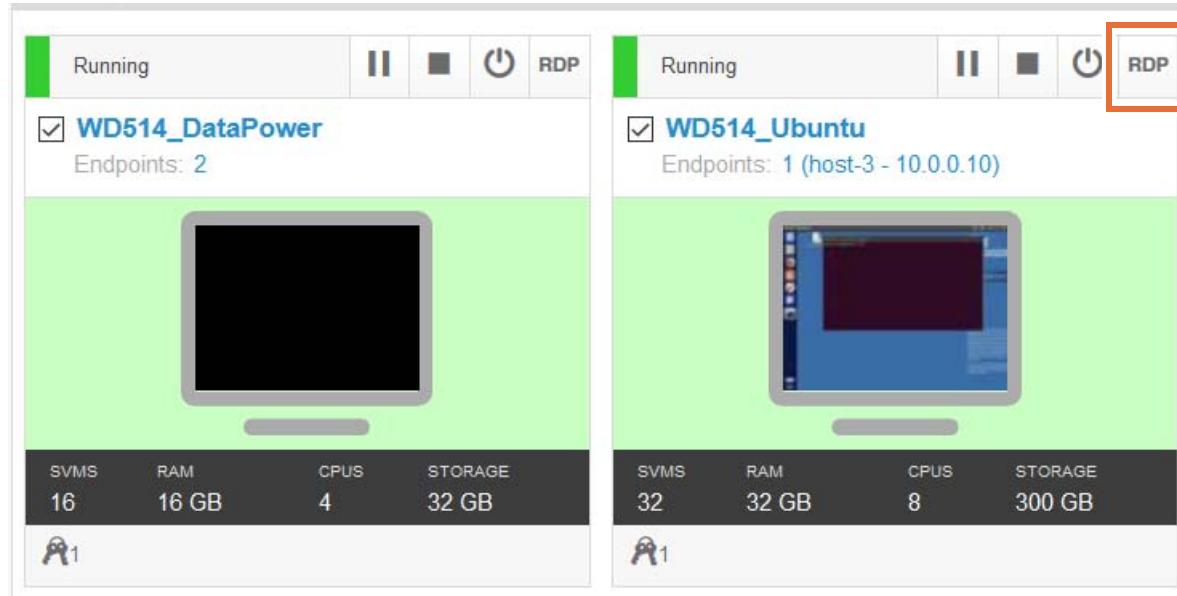
for the suspended VMs. When you start the VM after three hours of suspension, the clock is no longer paused and continues on from January 25, 10:00 AM. After running the VM for one hour it will then be January 25, 11:00 AM EST.

1.2.5. Accessing the virtual images through Remote Desktop

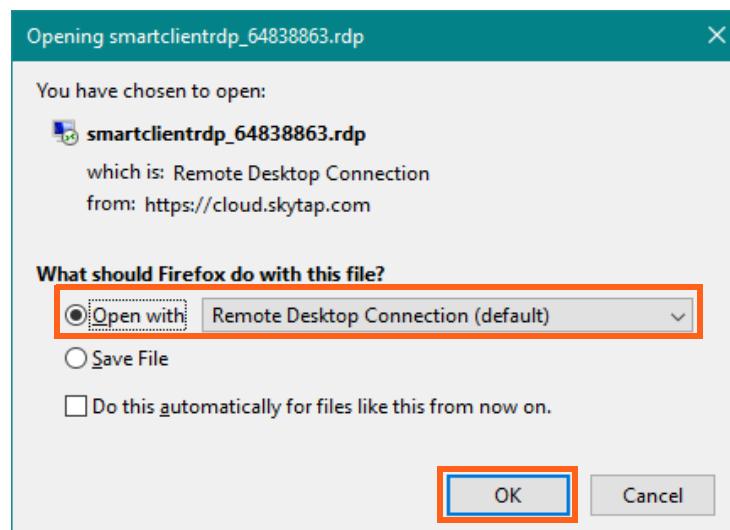
You can open the virtual machines in a web browser or by using a remote desktop connection.

To use a remote desktop connection, click **RDP** on the Ubuntu image. You can download the RDP file or open it with Remote Desktop on your workstation. To do so,

- 1. After both VMs are started, click **RDP** on the WD514_Ubuntu VM..



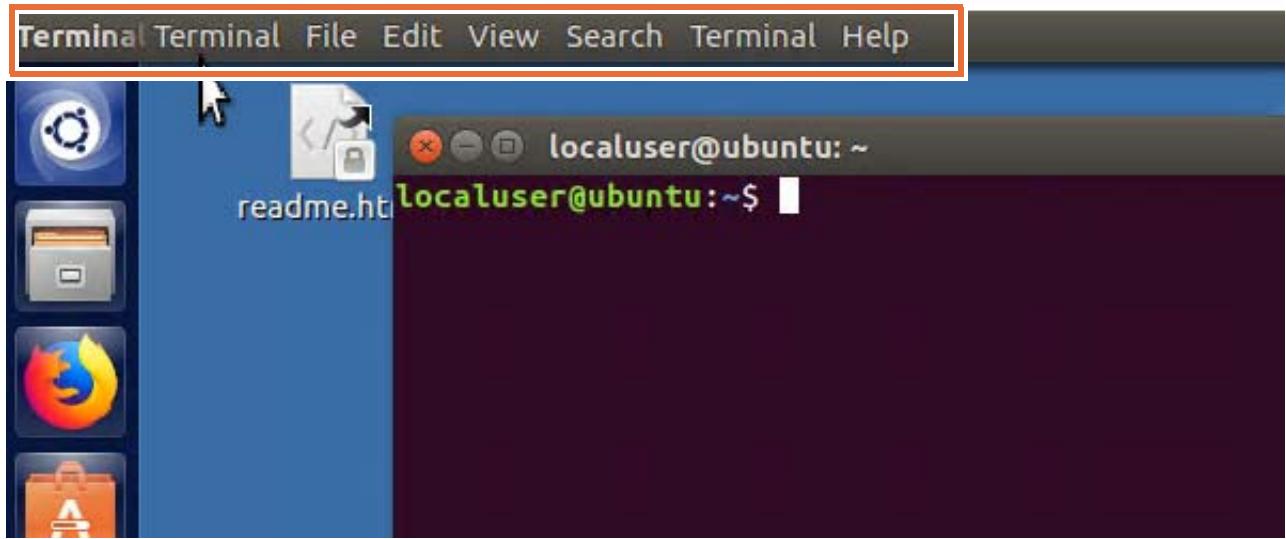
- 2. Leave the option to Open with **Remote Desktop Connection (default)** and click **OK**.



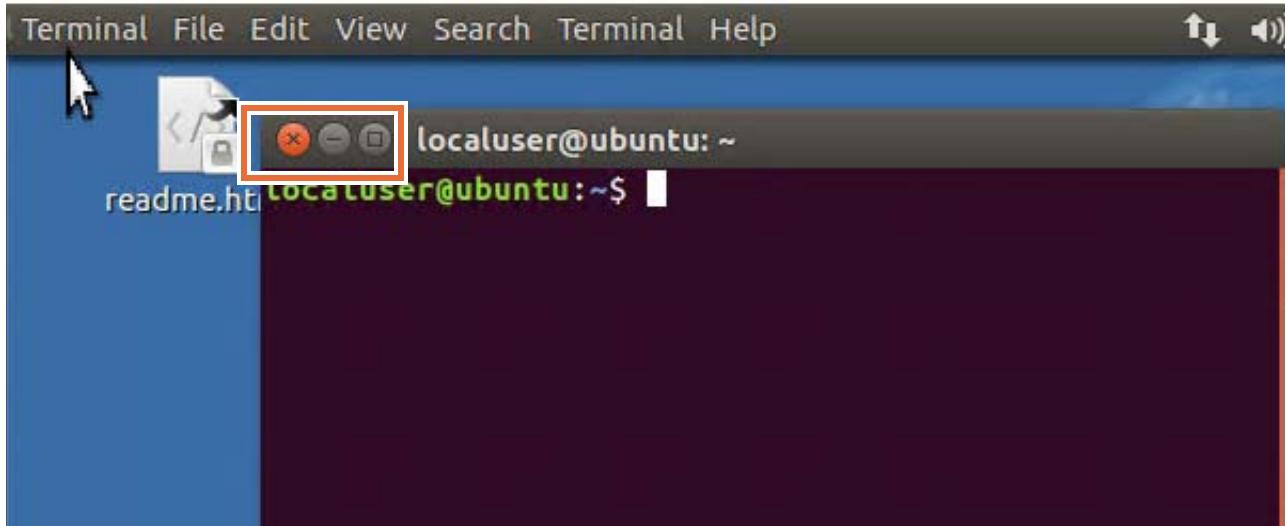
The session opens in Remote Desktop.

1.2.6. Working with the Unity desktop

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 mouse 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.

Exercise 1. Reviewing the API Connect development and runtime environment

Estimated time

01:00

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 in as the administrator to the Cloud Manager user interface and review the services that are defined in the Cloud Manager.

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

Before starting the exercises, **you must read** "[Before you begin](#)" on page 1-3 to understand Kubernetes certificates, how date and time settings affect the virtual machines, and how to start, suspend, and restart the virtual images that are provided for your lab environment.

Exercise instructions

This exercise provides 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 during the course.



Attention

It is critical that you read the "[Before you begin](#)" on page 1-3. Be sure to follow the instructions to access, suspend and restart your lab environment.

1.1. Starting your lab environment

This section ensures that you can access and manage the virtual machines in your lab environment. Your lab environment includes two virtual images:

- DataPower
- Ubuntu

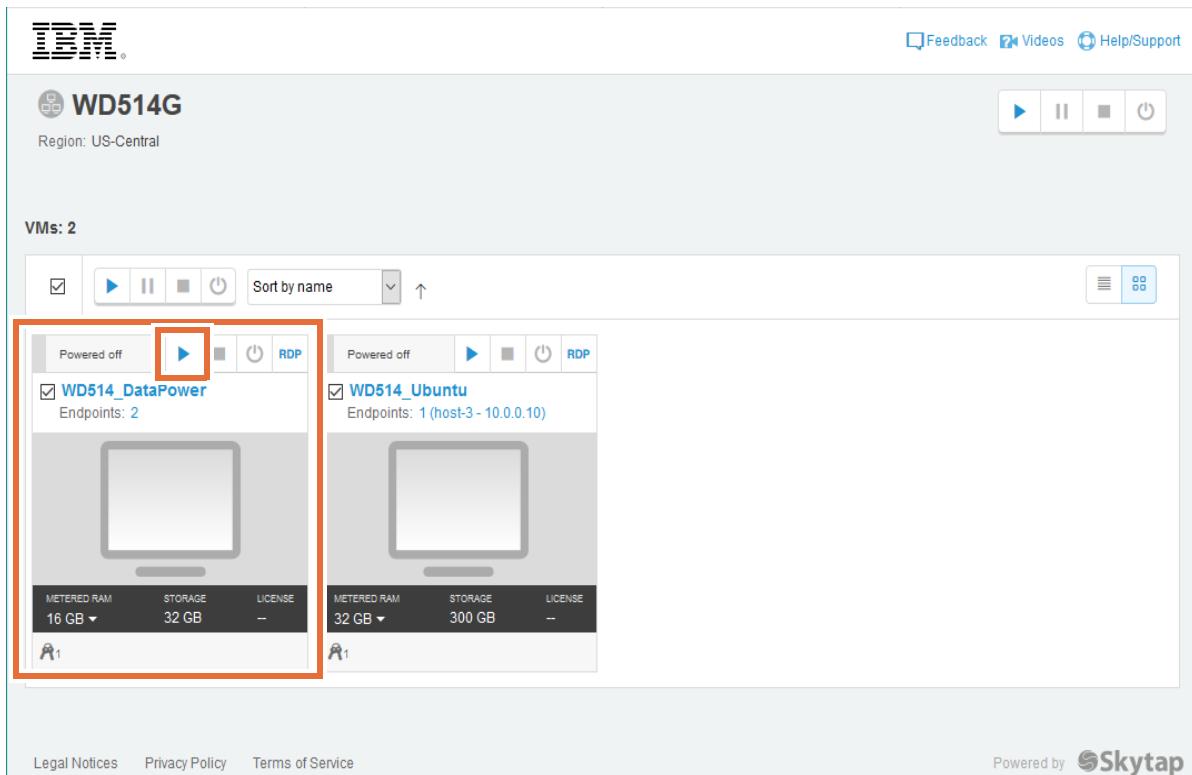
1.1. To manage the virtual images in your lab environment

- 1. Before you start working in the lab environment, read "[Before you begin](#)" on page 1-3.
- 2. **Do not** change the date or time on the Ubuntu image.
See "[Working with Kubernetes certificates](#)" on page 1-3.
- 3. **Do not** shut down the Ubuntu image after you start it.
See "[Working with Kubernetes certificates](#)" on page 1-3.

1.2. To start the virtual images

You start the DataPower image first. The DataPower image must be running before you start the Ubuntu image.

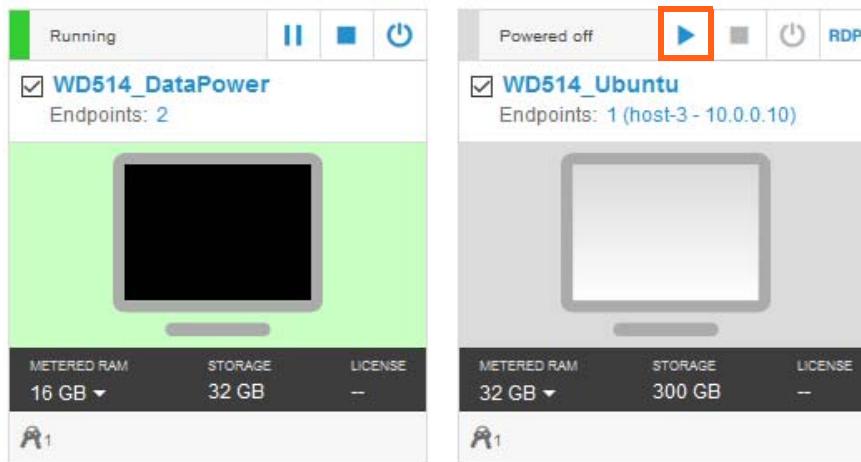
- 1. Start the DataPower virtual image.
 - a. Click the **Run this VM** icon for the WD514_DataPower image.



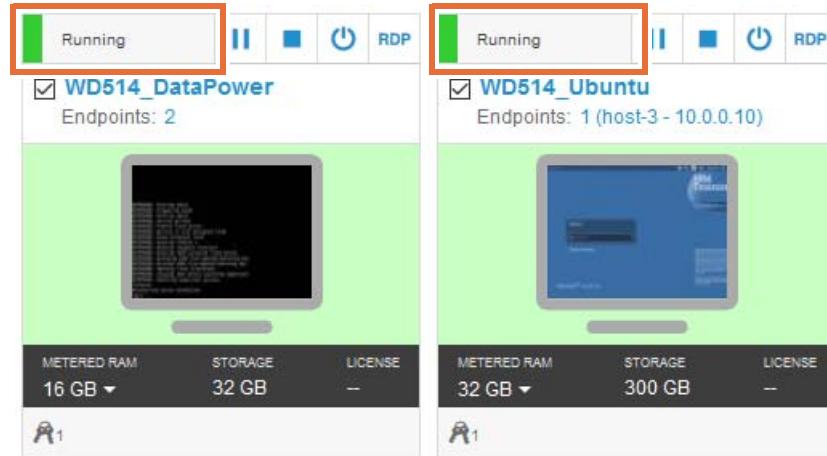
- b. Wait for the status to change to **Running**.

— 2. **After the DataPower image is running**, start the Ubuntu image.

— a. Click the **Run this VM** icon for the UbuD514_Ubuntu image.



— b. Wait for the status to change to **Running**.



— 3. Wait 10 minutes to allow the synchronization to complete and for all pods to start.



Important

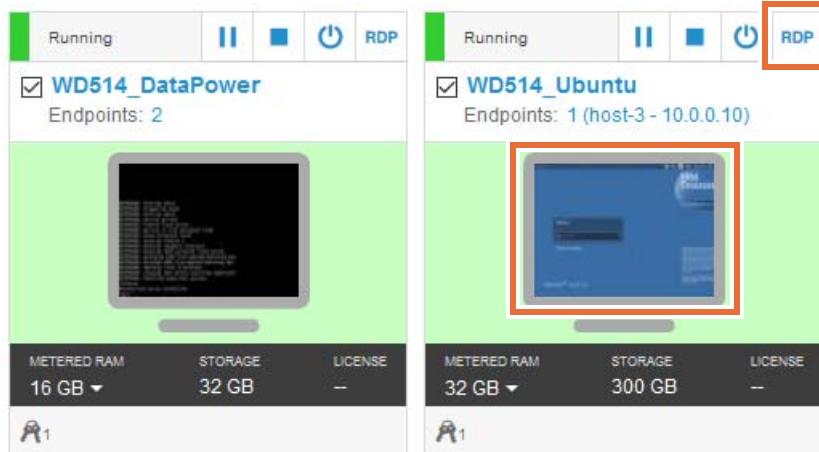
Do not shut down the WD514_Ubuntu VM after starting it.

1.3. To access the Ubuntu image

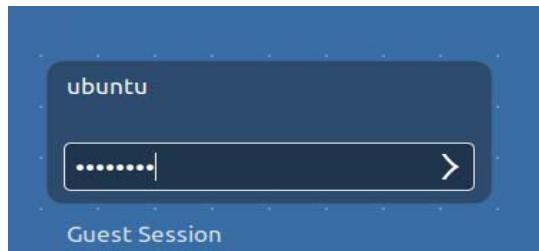


Information

You can either click the desktop image to open in a browser or click **RDP** to open in Remote Desktop. See "[Accessing the virtual images through Remote Desktop](#)" on page 1-8.



- ___ 1. Access the Ubuntu image by clicking the desktop.
- ___ 2. Log in to the Ubuntu VM with the password: `passw0rd`



Information

The Ubuntu student workstation opens to the desktop. The Ubuntu credentials are:

- User: `localuser`
- Password: `passw0rd`



Attention

To suspend your lab environment at the end of the day and to start the next day, make sure that you follow the instructions in "[Before you begin](#)" on page 1-3.

1.2. 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`
You see the following results.

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

Non-authoritative answer:
Name: google.com
Address: 172.217.2.238



Note

The IP Address might be different when you run the command.

- ___ c. In the terminal, type nslookup cloud.think.ibm.
You see the following results.

```
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
```

1.3. Review the Kubernetes certificates

For this course, IBM API Connect runs on a Kubernetes environment, sometimes abbreviated as K8. As mentioned in the **Before you begin** section, client certificates generated by `kubeadm` expire after 1 year by default. At the time this course was created, certificate rotation was not implemented. Due to certificate expiration, the date in the environment is set to January 25, 2020.

Enter the date command to check the date of the Virtual Machine (VM) when you first access the machine. The actual date and time varies depending on how long the VM has been running.

- ___ 1. Verify the system date. In the terminal, type `date`.

```
localuser@ubuntu:~$ date
Sat Jan 25 11:53:57 EST 2020
```

- ___ 2. Verify Kubernetes certificate expiration dates.

- ___ a. To view the current expiration date on the x509 certificate, enter the following command:

```
openssl x509 -in /etc/kubernetes/pki/apiserver.crt -noout -text |grep 'Not '
```

The result is displayed.

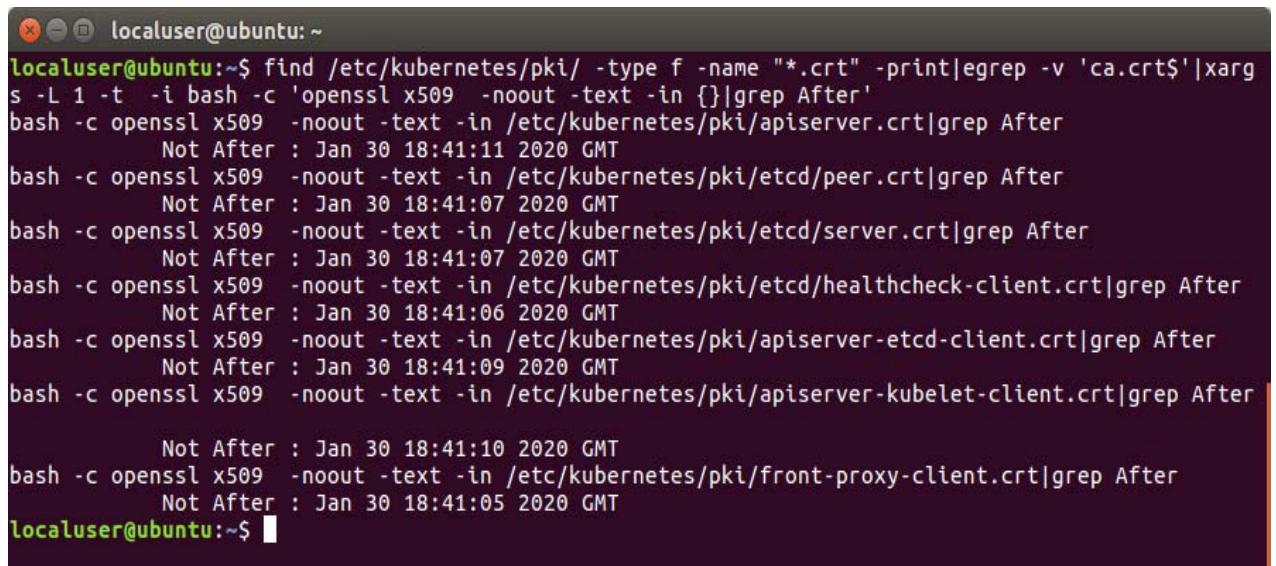
```
Not Before: Jan 30 18:41:10 2019 GMT
Not After : Jan 30 18:41:11 2020 GMT
```

Notice that the system date is within the window of the x509 certificate.

- ___ b. To check all certificate expiration dates, enter the following command:

```
find /etc/kubernetes/pki/ -type f -name "*.crt" -print|egrep -v 'ca.crt$'|xargs -L 1 -t -i bash -c 'openssl x509 -noout -text -in {}|grep After'
```

The result is displayed.



```
localuser@ubuntu:~$ find /etc/kubernetes/pki/ -type f -name "*.crt" -print|egrep -v 'ca.crt$'|xargs -L 1 -t -i bash -c 'openssl x509 -noout -text -in {}|grep After'
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/apiserver.crt|grep After
    Not After : Jan 30 18:41:11 2020 GMT
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/etcd/peer.crt|grep After
    Not After : Jan 30 18:41:07 2020 GMT
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/etcd/server.crt|grep After
    Not After : Jan 30 18:41:07 2020 GMT
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/etcd/healthcheck-client.crt|grep After
    Not After : Jan 30 18:41:06 2020 GMT
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/apiserver-etcd-client.crt|grep After
    Not After : Jan 30 18:41:09 2020 GMT
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/apiserver-kubelet-client.crt|grep After
    Not After : Jan 30 18:41:10 2020 GMT
bash -c openssl x509 -noout -text -in /etc/kubernetes/pki/front-proxy-client.crt|grep After
    Not After : Jan 30 18:41:05 2020 GMT
localuser@ubuntu:~$
```

Notice that the system date is within the window of each certificate.



Important

Do not change the date or time on the Ubuntu VM. Doing so corrupts the environment. The date and time is configured in the **BIOS** settings for the Virtual Machines. Additionally, the VMs are configured not to get the current time from the internet. Do **NOT** change any of these date and time settings.

___ 3. Verify internet access.

Due to the system date being modified, access to external websites is limited. This will prevent you from accessing websites that use security certificates with expiration dates.

___ a. Open the browser and attempt access to the following site:

<http://www.ibm.com>

___ b. Accept the security exception from the web browser by clicking **Advanced...**

Warning: Potential Security Risk Ahead

Firefox detected a potential security threat and did not continue to apigw.think.ibm. If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

[Learn more...](#)

[Go Back \(Recommended\)](#)

[Advanced...](#)

Report errors like this to help Mozilla identify and block malicious sites

c. Click **Accept the Risk and Continue**.

Websites prove their identity via certificates. Firefox does not trust this site because it uses a certificate that is not valid for apigw.think.ibm. The certificate is only valid for .

Error code: MOZILLA_PKIX_ERROR_SELF_SIGNED_CERT

[View Certificate](#)

[Go Back \(Recommended\)](#) [Accept the Risk and Continue](#)

__ d. Verify that the connection failed. The browser returns the following response:

The screenshot shows a Mozilla Firefox browser window with the title bar "Problem loading page - Mozilla Firefox". The address bar displays "https://www.ibm.com". Below the address bar, the navigation menu includes "Most Visited", "Getting Started", "Cloud Manager", "API Manager", and "API Developer Portal". The main content area features a large heading "Secure Connection Failed" followed by a descriptive message: "An error occurred during a connection to www.ibm.com. The OCSP response is not yet valid (contains a date in the future). Error code: SEC_ERROR_OCSP_FUTURE_RESPONSE". A bulleted list provides two solutions: "The page you are trying to view cannot be shown because the authenticity of the received data could not be verified." and "Please contact the website owners to inform them of this problem." At the bottom left, there is a link "Learn more..." and a checkbox for reporting errors. On the bottom right, a blue button says "Try Again".

e. Close the browser.



Note

Due to the system date issue, you cannot access the IBM Knowledge Center from inside the Ubuntu VM.

1.4. Review the Kubernetes runtime environment

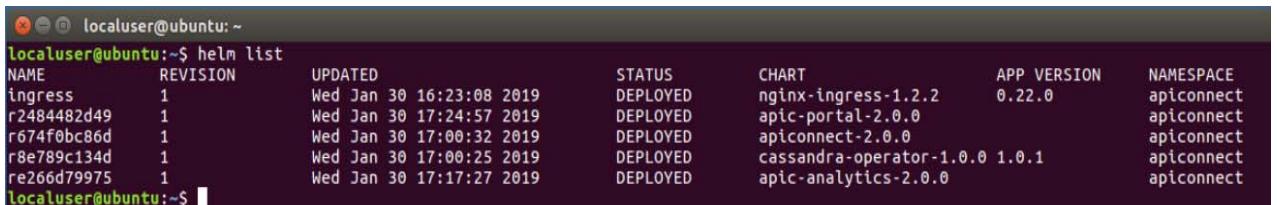
The Kubernetes environment supports scalability and failover. For this course, Kubernetes is set up with a single master node and all the processes run on the same virtual machine. This configuration is not scalable and is used only for demonstration purposes. Kubernetes manages the Docker containers that provide the runtime environment. These components start automatically when the student image is started. You 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. Return to the terminal window.
- b. Type `helm list`.

The list of deployed helm charts is displayed.



```
localuser@ubuntu:~$ helm list
NAME          REVISION        UPDATED         STATUS        CHART
ingress       1               Wed Jan 30 16:23:08 2019  DEPLOYED    nginx-ingress-1.2.2
r2484482d49  1               Wed Jan 30 17:24:57 2019  DEPLOYED    apic-portal-2.0.0
r674f0bc86d  1               Wed Jan 30 17:00:32 2019  DEPLOYED    apiconnect-2.0.0
r8e789c134d  1               Wed Jan 30 17:00:25 2019  DEPLOYED    cassandra-operator-1.0.0
re266d79975  1               Wed Jan 30 17:17:27 2019  DEPLOYED    apic-analytics-2.0.0
localuser@ubuntu:~$
```



Troubleshooting

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`

The list of pods is displayed.

NAME	READY	STATUS	RESTARTS	AGE
hostpath-provisioner-77d68bd579-vgqjq	1/1	Running	31	359d
ingress-nginx-ingress-controller-58frc	1/1	Running	59	359d
ingress-nginx-ingress-default-backend-7f7bf55777-7rzc8	1/1	Running	32	359d
r2484482d49-apic-portal-db-0	2/2	Running	65	359d
r2484482d49-apic-portal-nginx-76c946fdbb-6ldht	1/1	Running	32	359d
r2484482d49-apic-portal-www-0	2/2	Running	92	359d
r674f0bc86d-a7s-proxy-765d77f6b6-5gdvw	1/1	Running	28	359d
r674f0bc86d-apiconnect-cc-0	1/1	Running	32	359d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549470600-42jkg	0/1	Error	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549470600-cdpqj	0/1	Error	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549470600-dkvtf	0/1	Error	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549470600-gs2df	0/1	Error	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549470600-hzm28	0/1	Error	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549470600-zl8sw	0/1	Error	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549474200-m2542	0/1	Completed	0	353d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549488600-62nvf	0/1	Error	0	352d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549488600-bj6cv	0/1	Error	0	352d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549488600-pkgft	0/1	Error	0	352d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549488600-q9s4s	0/1	Error	0	352d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549488600-qtcjw	0/1	Completed	0	352d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549488600-vnck2	0/1	Error	0	352d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549578600-57f2c	0/1	Error	0	351d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549578600-76rrm	0/1	Error	0	351d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549578600-7h4qt	0/1	Error	0	351d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549578600-vdx6t	0/1	Completed	0	351d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549578600-xd5p2	0/1	Error	0	351d
r674f0bc86d-apiconnect-cc-cassandra-stats-1549672200-gbgjn	0/1	ContainerCreating	0	350d
r674f0bc86d-apiconnect-cc-repair-1549674000-995mg	0/1	ContainerCreating	0	350d
r674f0bc86d-apiconnect-cc-repair-1554944400-4bctd	0/1	Error	0	289d
r674f0bc86d-apiconnect-cc-repair-1554944400-9xxss	0/1	Error	0	289d
r674f0bc86d-apiconnect-cc-repair-1554944400-bch4n	0/1	Completed	0	277d
r674f0bc86d-apiconnect-cc-repair-1554944400-jqxkg	0/1	Error	0	289d
r674f0bc86d-apiconnect-cc-repair-1555808400-qtbjqj	0/1	Completed	0	277d
r674f0bc86d-apiconnect-cc-repair-1555981200-ds4wh	0/1	Completed	0	277d
r674f0bc86d-apim-schema-init-job-ltcpz	0/1	Completed	0	359d
r674f0bc86d-apim-v2-88674dd9f-g8r95	1/1	Running	34	359d
r674f0bc86d-client-dl-srv-b7fdf9767-6g4qz	1/1	Running	32	359d

- ___ b. 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.

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
```

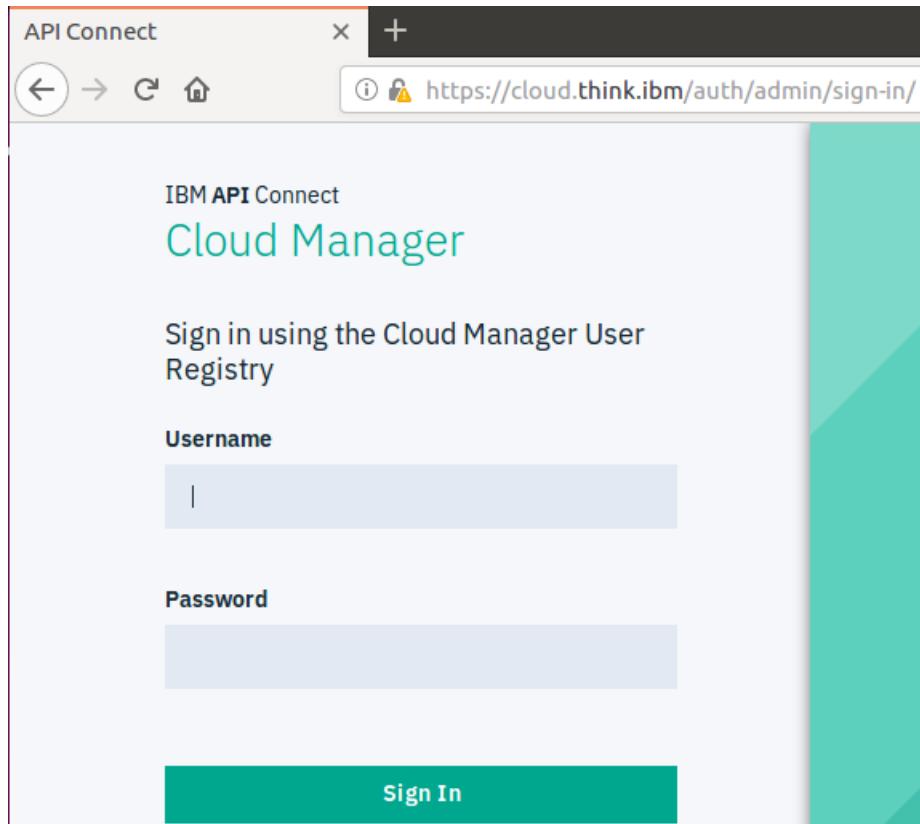
The output from running this command shows a master node and the Kubernetes version.

```
localuser@ubuntu:~$ kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS   AGE
calico-etcd-x58fk                 1/1     Running   30          305d
calico-kube-controllers-89d649f4f-6c6k8   1/1     Running   36          305d
calico-node-rsbind                 1/1     Running   37          305d
coredns-86c58d9df4-chrpib          1/1     Running   29          305d
coredns-86c58d9df4-thqb2           1/1     Running   29          305d
etcd-ubuntu                         1/1     Running   42          305d
kube-apiserver-ubuntu              1/1     Running   39          305d
kube-controller-manager-ubuntu      1/1     Running   41          305d
kube-proxy-tq2pn                   1/1     Running   30          305d
kube-scheduler-ubuntu              1/1     Running   40          305d
```

1.5. 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 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:

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

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.



Note

The Service may be labeled API DataPower Gateway or you might see DataPower API Gateway.

___ 3. Review the provider organization.

___ a. Hover over the left navigation bar and click **Provider Organizations**.

The screenshot shows the IBM API Connect Cloud Manager interface. The left sidebar has a red box around the 'Provider Organizations' link. The main content area has three cards: 'Configure Cloud' (with a gear icon), 'Configure Topology' (with a network icon), and 'Learn more' (with a book icon). The 'Learn more' card includes the text 'Documentation and tutorials with step-by-step instructions'.

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

			Add ▾
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.

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.

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.

The admin user was created in the Cloud Manager local user registry during product installation.

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 **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.

In many circumstances, users are created with the Developer role that can be used for creating and editing APIs.

- ___ d. Expand **Owner** to see the permissions of the owner of the provider organization. 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.

Owner		
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	Drafts	Product
<ul style="list-style-type: none"> • View 	<ul style="list-style-type: none"> • View • Edit 	<ul style="list-style-type: none"> • View • Stage • Manage
Product-Approval	Consumer-Org	App
<ul style="list-style-type: none"> • View • Stage • Publish • Supersede • Replace • Deprecate • Retire • Archive 	<ul style="list-style-type: none"> • View • Manage 	<ul style="list-style-type: none"> • View • Manage
App-Dev	App-Approval	Subscription
<ul style="list-style-type: none"> • Manage 	<ul style="list-style-type: none"> • View • Manage 	<ul style="list-style-type: none"> • View • Manage

- e. Click **Endpoints** from the Settings page.

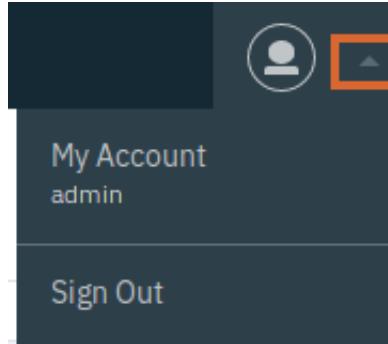
Settings

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 with an orange border), 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 in the upper right.



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.

Exercise 2. Creating an API definition from an existing API

Estimated time

01:30

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 POST 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 use the Product application to build an API. The sample Product application is located at: store has many operations pre-built that you can use to create APIs.

After you test the working API implementation, you design a set of API operations in the API Manager. You map the operations to the API implementation with an invoke message policy. Lastly, you test the API definition with the built-in test client of API Manager.

Requirements

You must complete this lab exercise in the Ubuntu host environment.

2.1. Review the Petstore REST API sample application

The **Swagger Petstore** REST API provides access to operations that allow a service to GET, POST, PUT, and DELETE entries in the Petstore database. The Swagger Petstore is a sample pet store server that developers can use to test their services. It also provides support to test authorization filters using an api key.

In this section, you review the operations of the Swagger Petstore 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.

— 1. Open the Swagger Petstore REST API in a browser.

— a. In the browser, type `http://petstore.swagger.io`

Swagger UI - Mozilla Firefox

Swagger UI

petstore.swagger.io

Most Visited Getting Started Cloud Manager API Manager API Developer Portal

Swagger. Supported by SMARTBEAR

<http://petstore.swagger.io/v2/swagger.json> Explore

Swagger Petstore 1.0.5

[Base URL: petstore.swagger.io/v2]
<http://petstore.swagger.io/v2/swagger.json>

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net, #swagger](#). For this sample, you can use the api key **special-key** to test the authorization filters.

[Terms of service](#)
[Contact the developer](#)
[Apache 2.0](#)
[Find out more about Swagger](#)

Schemes **HTTP**

Authorize

pet Everything about your Pets

Find out more: <http://swagger.io>



Troubleshooting

If the petstore fails to load due attempting to load the secured version (https) of the page, replace the **https** portion of the URL with **http**.

Failed to load API definition.

Errors

Fetch error
NetworkError when attempting to fetch resource. <https://petstore.swagger.io/v2/swagger.json>

Fetch error
Possible cross-origin (CORS) issue? The URL origin (<https://petstore.swagger.io>) does not match the page (<http://petstore.swagger.io>). Check the server returns the correct 'Access-Control-Allow-*' headers.

Hide

Due to the system date being set, you cannot access the secured version of the petstore API.

- ___ b. In a separate browser tab, view the OpenAPI specification for the Petstore server by using this URL:

http://petstore.swagger.io/v2/swagger.json

```

swagger: "2.0"
info:
  description: "This is a sample server Petstore server. You can [http://swagger.io](http://swagger.io) or on [irc://irc.mozilla.org](irc://irc.mozilla.org). For this sample, you can use the api key filters."
  version: "1.0.2"
  title: "Swagger Petstore"
  termsOfService: "http://swagger.io/terms/"
  contact:
    email: "apiteam@swagger.io"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
  host: "petstore.swagger.io"
  basePath: "/v2"
tags:
  0:
    name: "pet"
    description: "Everything about your Pets"
    externalDocs:
      description: "Find out more"
      url: "http://swagger.io"
  1:
    name: "store"
    description: "Access to Petstore orders"

```

- ___ c. Review the JSON, Raw Data, and Headers.
- ___ 2. Review the Pet Model.
 - ___ a. Return to the tab with the Swagger Petstore. Scroll to the bottom of the Petstore page to view the Models.

- ___ b. Click the **Pet** model.

```
Pet ↴ {  
    id  
    category  
    name*  
    photoUrls*  
    tags  
    status  
  
        integer($int64)  
        Category > {...}  
        string  
        example: doggie  
        > [...]  
        > [...]  
        string  
  
        pet status in the store  
  
        Enum:  
        > Array [ 3 ]  
}
```

The Pet schema contains several parameters. For the purpose of this exercise, the parameters **id** and **name** are used to identify a specific pet.

- ___ 3. Review and test the POST API operation

- ___ a. Scroll to the **pet** section at the top of the page, and click **POST /pet**.

- ___ b. Review the operation. This operation uses the pet object as a parameter.

POST /pet Add a new pet to the store

Parameters

Name **Description**

body * required	Pet object that needs to be added to the store
object (body)	Example Value Model

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Parameter content type

application/json ▾

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

- ___ d. Enter a unique **name** and pet **id** as part of the pet object to pass.

```
{
  "id": 98711,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie98711",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

- ___ e. Click **Execute**.
- ___ f. Review the results.
 - A curl command representing the same web call is displayed.
 - Request URL is displayed.
 - Server response is displayed. The response contains the pet object.



Information

The cURL utility is a widely customizable, third-party application for HTTP network testing. In this course, you use a combination of command line utilities and web application clients to test API operations.

- ___ 4. Review and test the GET API operation
 - ___ a. Scroll to the **GET /pet/{petId}** operation and click **GET**.

Name	Description
petId * required integer(\$int64) (path)	ID of pet to return

petId - ID of pet to return

This operation uses the **petId** as a parameter and returns the **Pet** object.

You use this path when creating the GET operation.

Note that the petId variable type is integer (int64).

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

- ___ c. In the **ID of pet to return** field, type the pet ID you used in your POST operation.

The screenshot shows a user interface for executing a REST API call. At the top, it displays a blue button labeled "GET" and a URL path "/pet/{petId} Find pet by ID". To the right of the URL is a small padlock icon. Below the URL, the text "Returns a single pet" is displayed. On the left side, there is a section titled "Parameters" with a red "Cancel" button on the right. A table lists a parameter named "petId * required" with the description "ID of pet to return (path)". The value "98711" is entered into the input field next to the parameter. At the bottom of the interface is a large blue button labeled "Execute".

- ___ d. Select `application/xml` for the **Response content type**.
___ e. Click **Execute**.

f. Review the results

- A curl command representing the same web call is displayed.
- Request URL is displayed.
- Server response is displayed. The response contains the pet object with the name of the pet retrieved.

Code	Details
200	Response body <pre><?xml version="1.0" encoding="UTF-8" standalone="yes"?> <Pet> <category> <id>0</id> <name>string</name> </category> <id>98711</id> <name>doggie98711</name> <photoUrls> <photoUrl>string</photoUrl> </photoUrls> <status>available</status> <tags> <tag> <id>0</id> <name>string</name> </tag> </tags> </Pet></pre> <div style="text-align: right;">Download</div>



Information

The Petstore is a publicly available web service. For purposes of this exercise, you create your own pet to use as test data.

To find a unique pet id, you can run the GET operation to query whether the id exists in the Petstore database. The operation can be run directly from a browser tab

(<http://petstore.swagger.io/v2/pet/98711>) or using the Swagger Petstore website. To do so, swap the id with an id you want to look up.

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

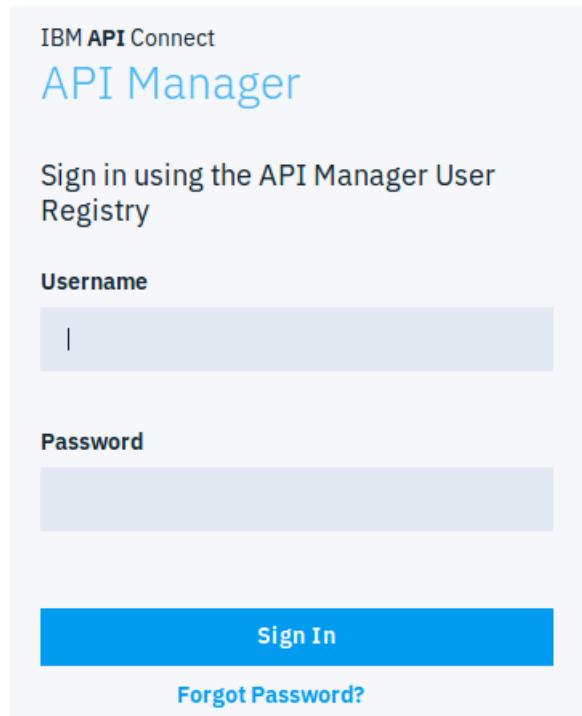
In this course, you use the API Manager browser-based application that runs on the student image for the development of APIs. The graphical interfaces to develop APIs and products are nearly identical in the API Designer and API Manager.

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 in to the API Manager development environment.

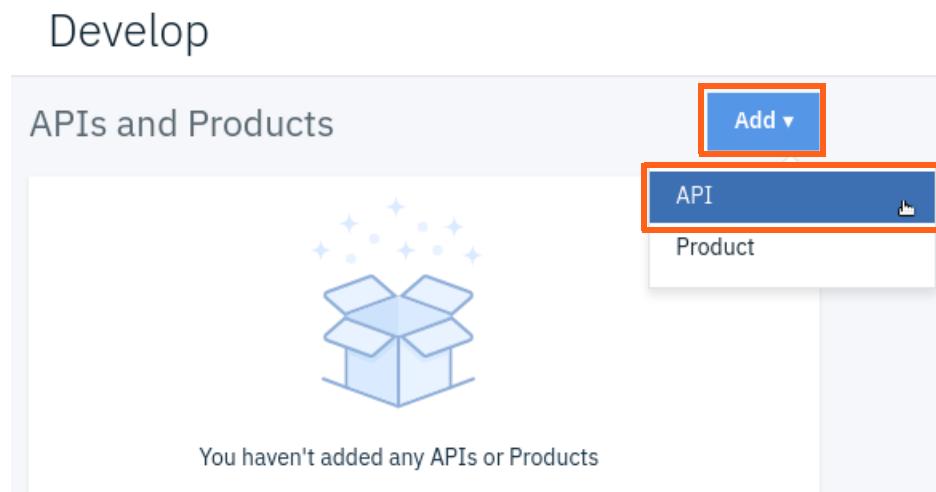
- a. In the browser, type the address:
`https://manager.think.ibm`

- ___ b. Sign in with the username and password that is associated with your API Manager account.

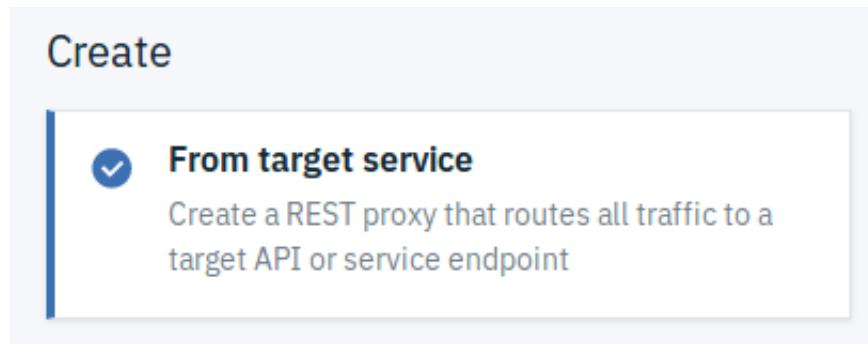


The screenshot shows the IBM API Connect API Manager sign-in page. At the top, it says "IBM API Connect" and "API Manager". Below that, it says "Sign in using the API Manager User Registry". There are two input fields: "Username" and "Password", both currently empty. Below the password field is a large blue "Sign In" button. Underneath the "Sign In" button is a link "Forgot Password?".

- Username: ThinkOwner
 - Password: Passw0rd!
- ___ c. Click **Sign in**.
- ___ 2. Configure the OpenAPI API definition.
- ___ a. From the API Manager home page, click the **Develop APIs and Products** tile.
- ___ b. From the Develop page, click the **Add > API**.



- __ c. Click the tile **From target service**, then click **Next**.



- __ d. Type the following properties for the API definition:

- Title: petstore
- Name: petstore
- Version: 1.0.0
- Base path: /v2
- Target service URL: <http://petstore.swagger.io/v2>

The dialog box contains the following fields:

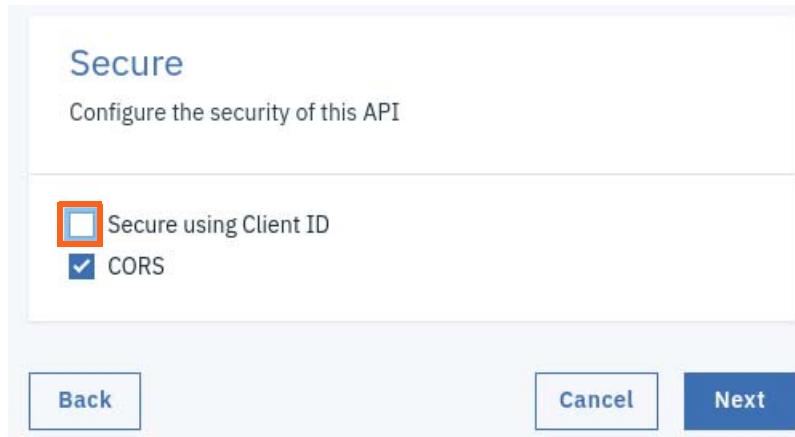
- Title:** petstore
- Name:** petstore
- Version:** 1.0.0
- Base path (optional):** /v2
- Target service URL:** <http://petstore.swagger.io/v2>

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

The base path **/v2** is the base path for the incoming API call, but since the out bound call needs **/v2**, you must add that to the target-url to use it in the invoke policy.

- __ e. Click **Next**.

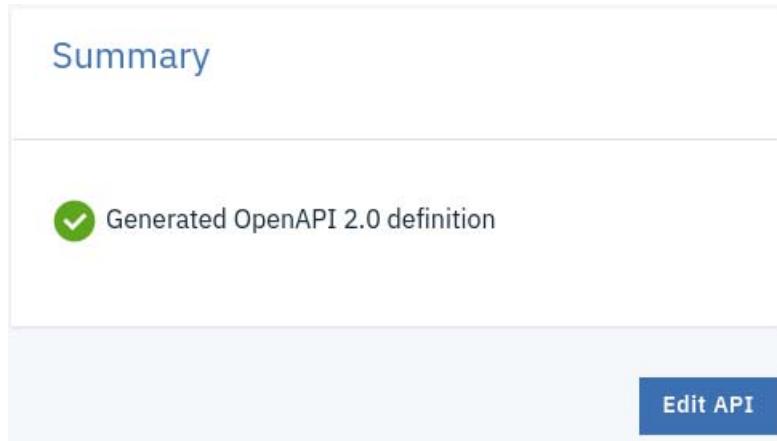
- __ f. Clear **Secure using Client ID** in the Secure dialog.



- __ g. Leave the **CORS** property selected.

- __ h. Click **Next**.

The OpenAPI 2.0 definition is generated.



- __ i. Click **Edit API**.

- __ 3. Examine the **petstore** API definition in the API Editor.

- __ a. The API Setup page opens in the Design view.

- ___ b. Review the API description metadata: Title, Name, and Version.

Develop

petstore 1.0.0

Design Source Assemble

API Setup

Security Definitions

Security

Paths

Definitions

Properties

Target Services

Categories

Activity Log

Info

Enter the API summary details

Title
petstore

Name
petstore

Version
1.0.0

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

Host

The host used to invoke the API

Address (optional)

Base Path

The base path is the initial URL segment of the API and does not include the any additional segments for paths or operations

Base path (optional)



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 hostname. For example, the address of the API Gateway is `http://apigw.think.ibm`. In this case, the entry point into the petstore API is `http://petstore.swagger.io/v2`.

- ___ d. Scroll down further and examine the **consumes** and **produces** sections.
- ___ e. 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)

Produces

Produces (optional)

- application/json
 application/xml

Add media type (optional)

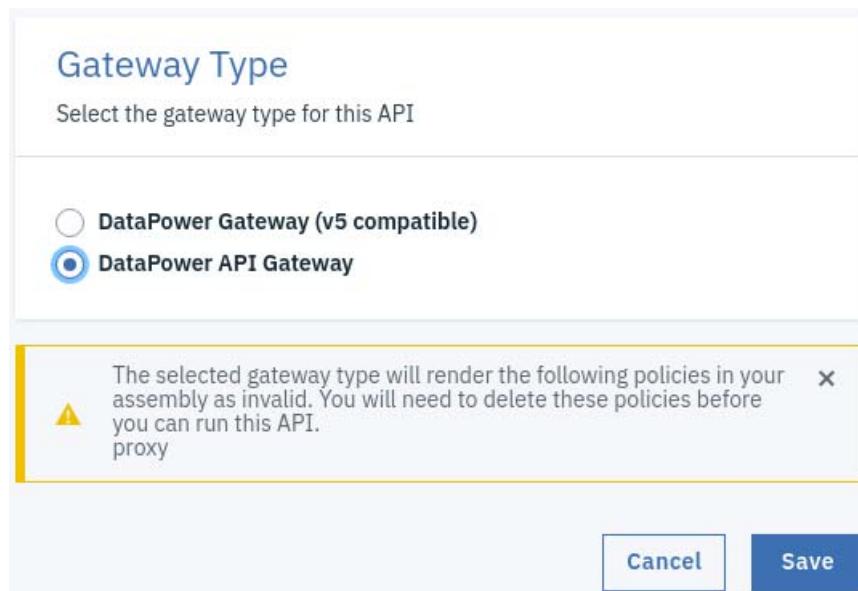


Information

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.



- c. Click **Save**.

A message that the API is updated appears in the upper right corner.



Note

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.

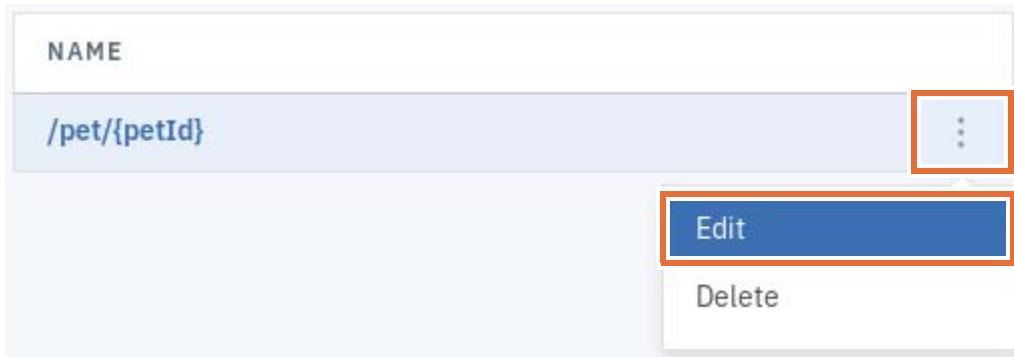
- 5. Create a path that is named `/pet/{petId}`.
 - a. Select the **Paths** section.

- ___ b. Click **Add** to create a path.
- ___ c. Change the path name to `/pet/{petId}`



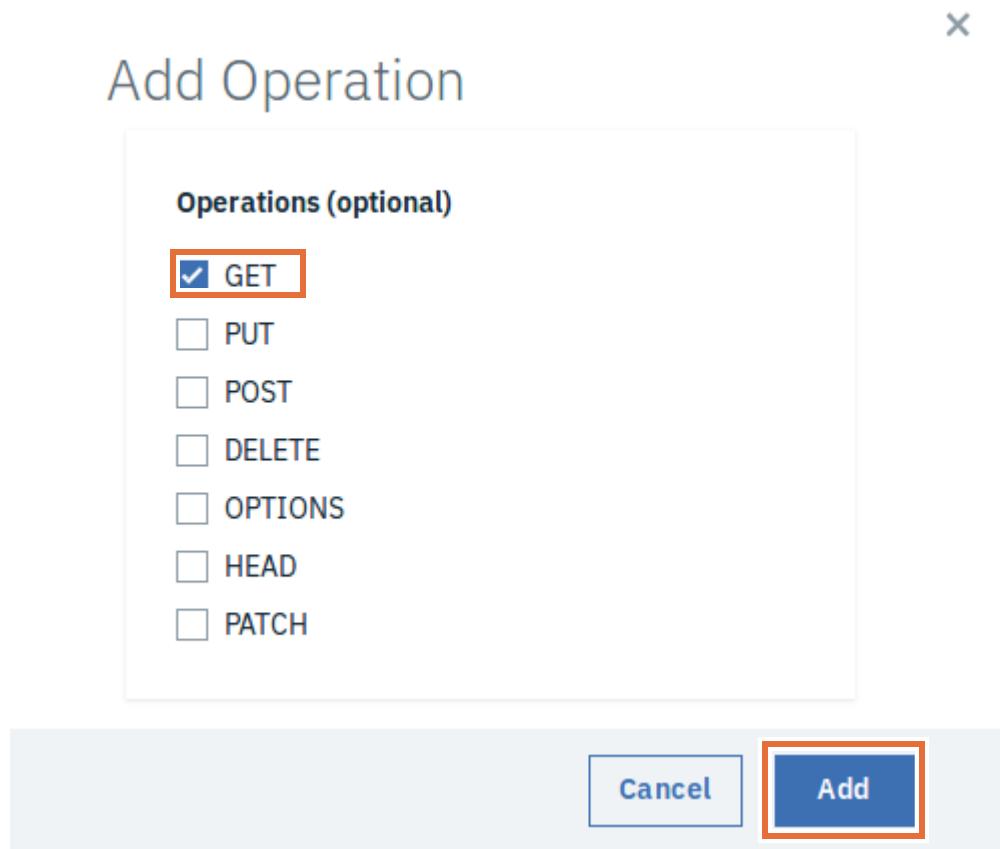
Recall this was the path observed in the GET operation on the Swagger Petstore website to retrieve a pet object by Id.

- ___ d. Click **Save**.
- ___ 6. Define a GET operation for the path.
 - ___ a. Click the vertical ellipsis three dots alongside the `/pet/{petId}` path and click **Edit**.



- ___ b. In the Operations section, click **Add**.

- __ c. Select GET and click **Add**.



- __ d. Click **Save**.
- __ 7. Add the parameter to the GET operation
- __ a. Click the vertical ellipsis (three dots) in the `/pet/{petId}` path and click **Edit**
- __ b. Click the vertical ellipsis (three dots) in the GET operation and click **Edit**.
- __ c. Enter `getPetById` for the **Operation Id**.
- __ d. Alongside the Parameters section, click **Add**.
- __ e. Complete the Parameter definition with the following values:
- REQUIRED:** checked
- NAME:** petId
- LOCATED IN:** path
- TYPE:** int 64

REQUIRED	NAME	LOCATED IN	TYPE	DESCRIPTION	DELETE
<input checked="" type="checkbox"/>	petId	path	int 64		

Recall that the **petId** was defined as an integer (int64) on the Swagger Petstore website.

- ___ 8. Add a response to the GET operation
 - ___ a. In the Response section, click **Add**.
 - ___ b. Complete the Response definition with the following:

STATUS CODE: 200

SCHEMA: object

DESCRIPTION: OK

STATUS CODE	SCHEMA	DESCRIPTION	DELETE
200	object ▾	OK	trash

- ___ c. Click **Save**.

2.3. Invoke a GET operation in the existing API implementation

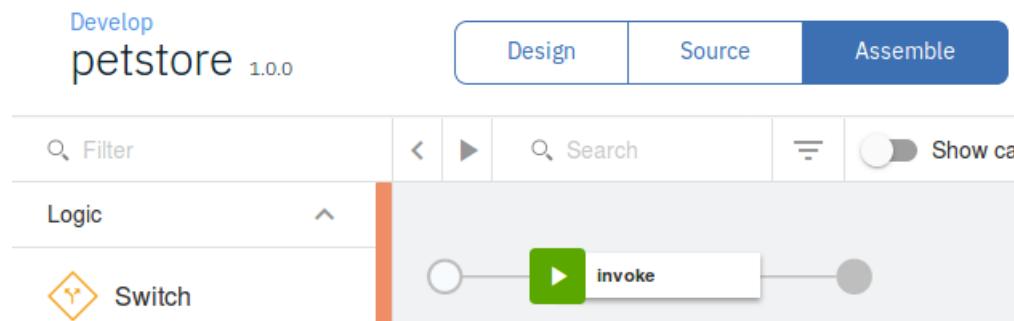
You created an API definition: the interface for the REST API. However, the interface describes the structure of the response message only. It does not implement the actual API operation.

In this section, you send the API operation that you defined in the API definition to the existing Swagger Petstore REST API application. To do this, you modify the **invoke** properties to call the `GET /getPetById` operation in the petstore API.

- 1. Open the **Assemble** view in the API editor.
 - a. In the petstore 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. Update the properties of the **invoke** policy.
 - a. Click the **invoke** policy in the sequence to open the properties view.

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



Information

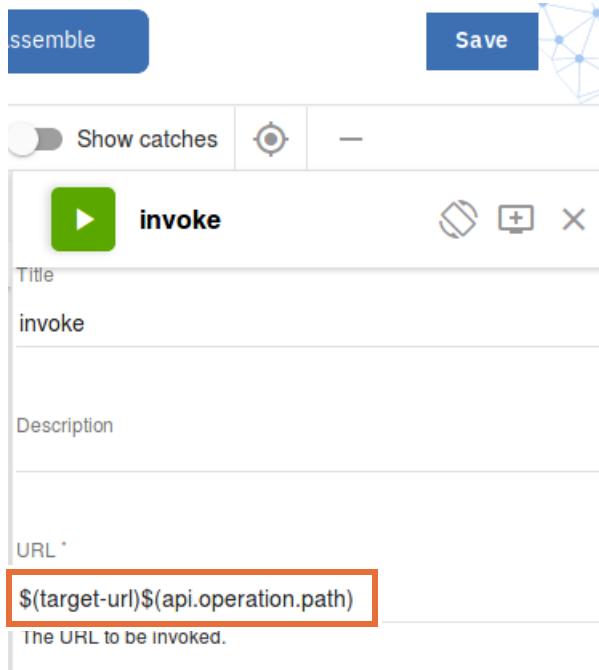
When the API Gateway receives a request for the `GET /getPetById` API operation, it routes the request to the message processing policy. The **invoke** operation calls the implementation of the API, at the address that is specified by the URL.

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 `$(target-url)$(api.operation.path)`. This is done in the next step.

The `target-url` is an API property that represents the hostname 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 `'/pet/{petId}'`, while the **request.path** property is `'/v2/pet/{petId}'`. The **request.path** property includes the base path.

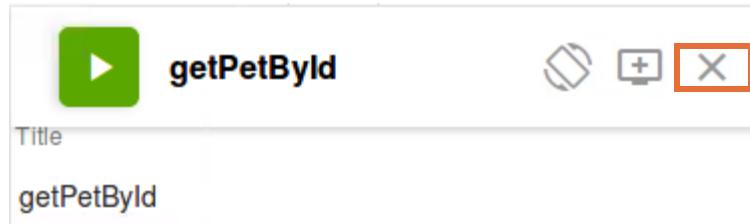
For more information, see the IBM Knowledge Center for API Connect and use the search string “API Connect context variables”.

Note: In the exercise, the target-url is `'http://petstore.swagger.io/v2'`.

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

The **api.operation.path** `/pet/{petId}` is concatenated with the target-url.

- ___ d. Change the **invoke** Title to `getPetById`
- ___ e. Click **Save**.
- ___ f. Close the properties view by clicking the **X** at the upper right.

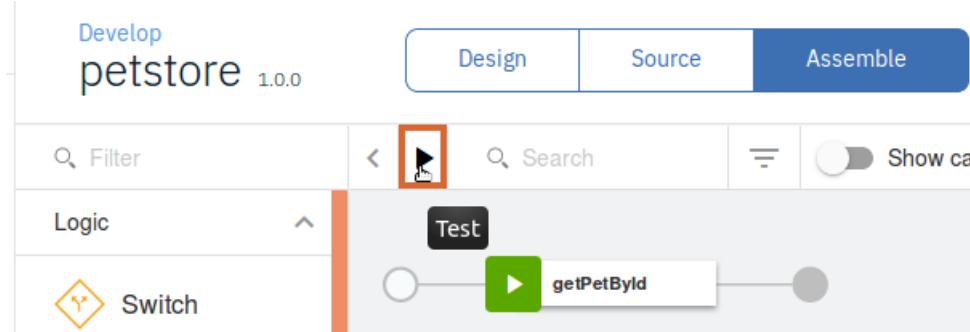


2.4. Test the API operation in the assembly

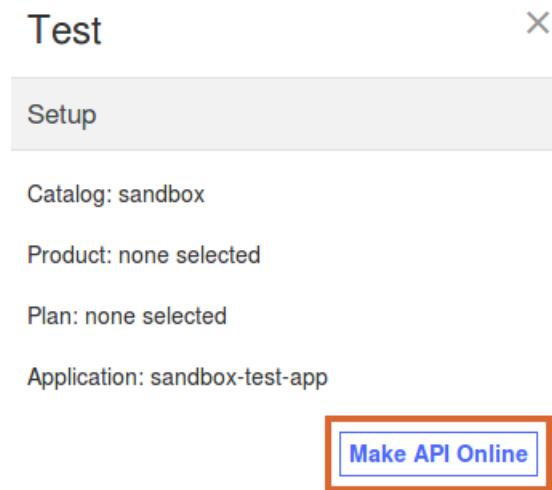
The API Manager includes a built-in test client for your API operations. In the **assemble** view, you can quickly invoke any operation in the API definition that you opened. For example, you can test the `GET /getPetById` operation from the petstore API.

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

- ___ 1. Test the `get /getPetById` 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 **Make API Online**.



This option automatically publishes the API to the sandbox catalog and makes it callable on the gateway. After the API is published, it can be republished if changes are made.

- ___ c. In the **Operation** section, select the get /pet/{petId} operation in the test client.

The screenshot shows the 'Test' interface with the 'Operation' section highlighted. A dropdown menu is open, showing the option 'get /pet/{petId}' which is highlighted with a red box.

- ___ d. Enter the pet ID you used earlier in the POST operation.

petId: 98711

- ___ 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.

The screenshot shows the 'Test' interface with the 'Generate' section highlighted. It includes fields for 'petId' (98711) and 'Repeat'. The 'Repeat' checkbox is unchecked, and the 'Stop on error' checkbox is checked.

- ___ f. Click **Invoke**.

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

Response
<p>Status code: -1</p> <p>No response received. Causes include a lack of CORS support on the target server, the server being unavailable, or an untrusted certificate being encountered.</p>
<p>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.</p> <p>https://apigw.think.ibm/think/sandbox/v2/pet/98798</p>
<p>Response time: 37ms</p>



Information

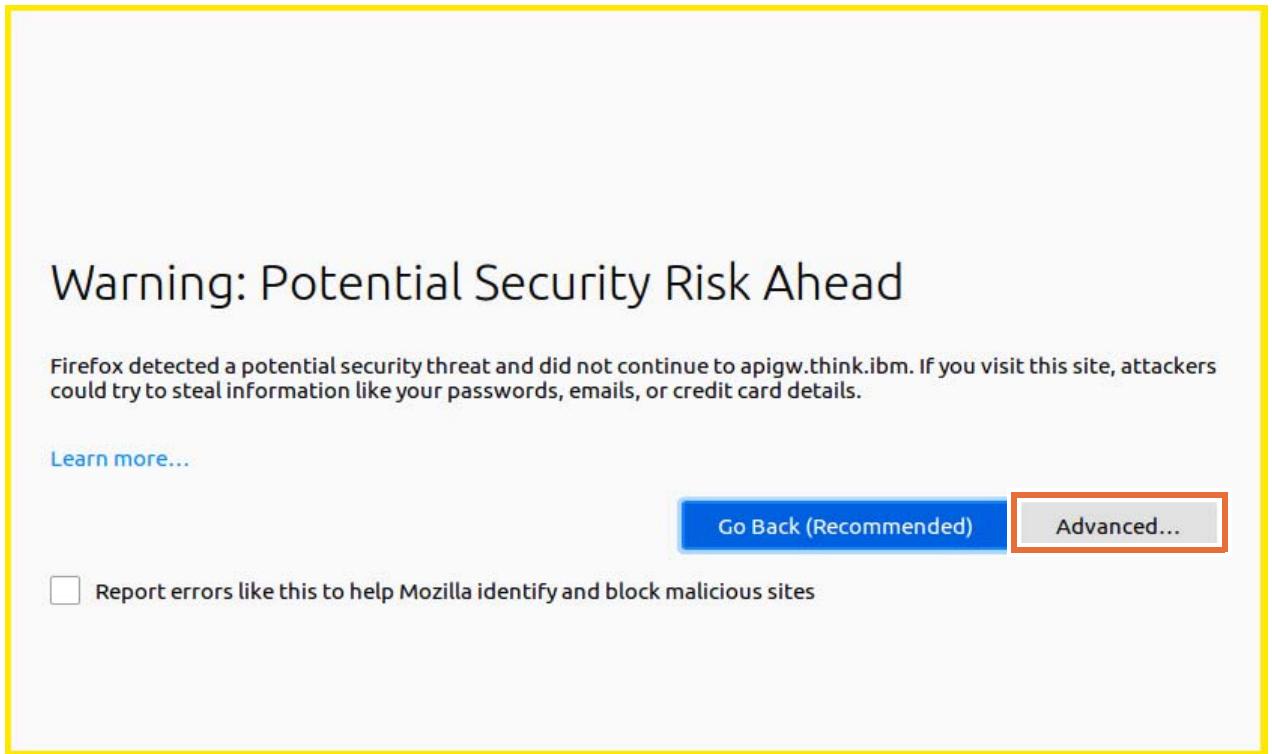
Why did the test client API call fail?

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.

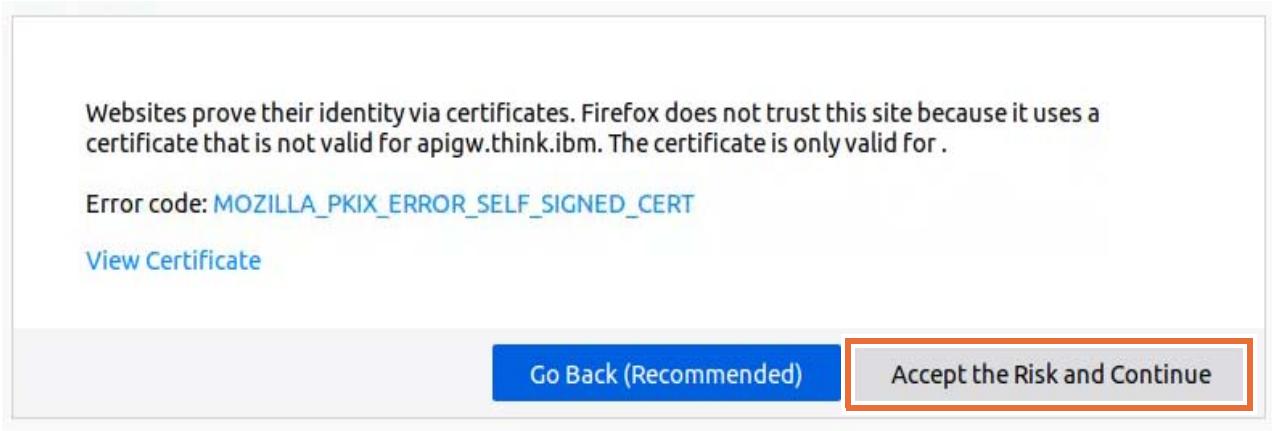
To fix this issue, you must create a security exception for the call to the gateway in your web browser. Accept the self-signed security certificate for the micro gateway with a security exception. This occurs only the first time you test an API.

- __ h. Click the link in the response area.

- __ i. Accept the security exception from the web browser by clicking **Advanced...**



- __ j. Click **Accept the Risk and Continue**.



- __ k. Examine the response from the API operation call.

- __ l. Verify in the Body, the **id** and **name** of the pet you added

```

- <Pet>
  - <category>
    <id>0</id>
    <name>string</name>
  </category>
  <id>98711</id>
  <name>doggie98711</name>
- <photoUrls>
  <photoUrl>string</photoUrl>
</photoUrls>
<status>available</status>
- <tags>
  - <tag>
    <id>0</id>
    <name>string</name>
  </tag>
</tags>
</Pet>
```

- __ m. Return to the tab with API Manager.
 __ n. Leaving the `petId` value, click **Invoke** again.
 __ o. This time the results are displayed in the Test client.

Body:

```
{
  "id": 98711,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie98711",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```



Troubleshooting

If you get an error when testing an API, this may be due to the two virtual machines being out of sync. Refer to the **Troubleshooting Issues** section in Appendix B.

To verify whether the environment is at issue or the API is incorrectly configured, you can import the solution for the petstore API here: [/lab_files/solutions/petstore_1.0.0.yaml](#). If this API receives a 200 status when invoked, this confirms that the issue is with the API configuration.

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

The Swagger Petstore REST API sample application also provides a `POST /pet` operation to add a pet. The operation accepts the pet object as a JSON object as seen on the Swagger Petstore website.

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

```
curl -X POST "http://petstore.swagger.io/v2/pet" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"id\": 98712, \"category\": { \"id\": 0, \"name\": \"string\" }, \"name\": \"doggie98712\", \"photoUrls\": [ \"string\" ], \"tags\": [ { \"id\": 0, \"name\": \"string\" } ], \"status\": \"available\"}"
```

In this section, you first define a pet object to use in the POST operation. Then, you create an API operation that is named `POST /pet` in your petstore OpenAPI definition. Then, you add an **Invoke** policy to make a POST request to the API implementation at `http://petstore.swagger.io`.

1. Define the pet object for the POST operation.

a. Recall the structure of the pet object from the Swagger Petstore

```
{
  "id": 1234,
  "category": {
    "id": 1234,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 1234,
      "name": "string"
    }
  ],
  "status": "available"
}
```

For the purposes of this exercise, you only use the **id** and **name** parameters.

- b. In the petstore API definition, switch to the Design view.
- c. In the left navigation menu, click **Definitions**.
- d. Click **Add** to create a schema definition.
- e. Set the name of the schema definition type to `pet`.

- ___ f. In the Properties section, click **Add** twice.
- ___ g. Build the properties based on the following values:

Table 1. Pet schema type definition

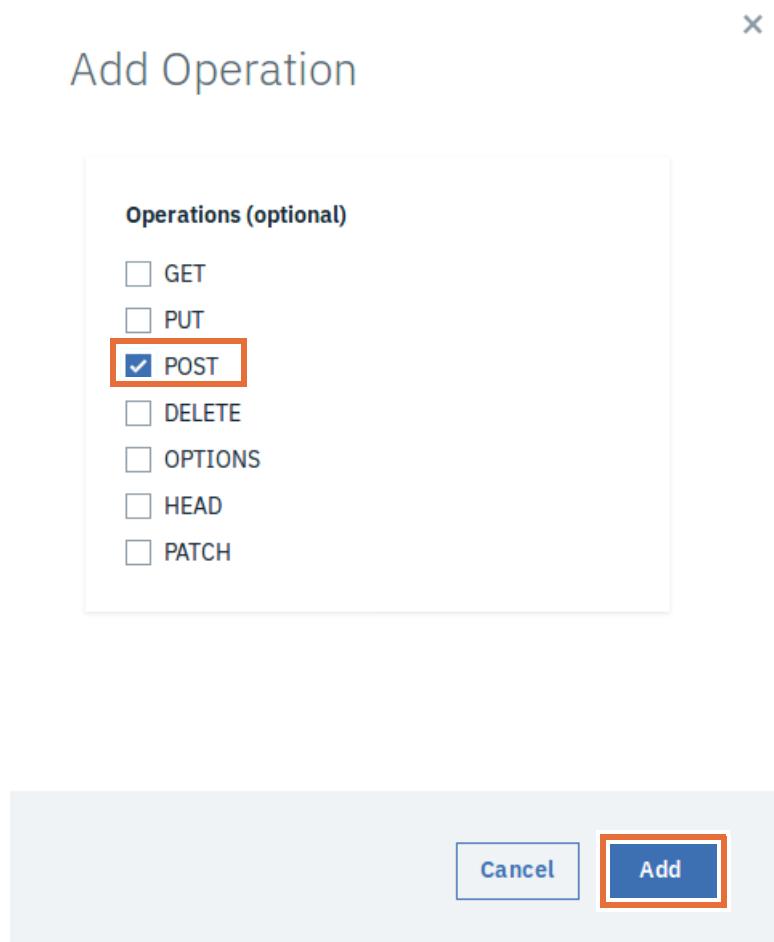
Properties name	Properties type	Properties example	Properties Description
id	integer	1234	pet id
name	string	string	pet name

- ___ h. Verify your properties.

PROPERTIES NAME	PROPERTIES TYPE	PROPERTIES EXAMPLE	PROPERTIES DESCRIPTION	DELETE
id	integer ▾	1234	pet id	
name	string ▾	string	pet name	

- ___ i. Click **Save**.
- ___ 2. Create a path that is named /pet.
 - ___ a. Select the **Paths** section.
 - ___ b. Click **Add** to create a path.
 - ___ c. Set the **Path name** to /pet
Recall this was the path observed in the POST operation on the Swagger Petstore website to add a pet.
 - ___ d. Click **Save**.
- ___ 3. Define a POST operation for the path.
 - ___ a. Click the vertical ellipsis (three dots) in the /pet path and click **Edit**.
 - ___ b. In the Operations section, click **Add**.

- __ c. Select POST and click **Add**.



The POST operation is added.

- __ d. Click **Save**.
- __ 4. Add a parameter to the POST operation
- __ a. Click the vertical ellipsis (three dots) in the **/pet** path and click **Edit**
 - __ b. Click the vertical ellipsis (three dots) in the POST operation and click **Edit**.
 - __ c. Enter `addPet` for the **Operation Id**.
 - __ d. In the Parameters section, click **Add**.
 - __ e. Complete the parameter definition with the following:
REQUIRED: checked
NAME: pet
LOCATED IN: body
TYPE: pet

DESCRIPTION: pet object

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

- ___ 5. Add a response to the POST operation
 - ___ a. in the Response section, click **Add**.
 - ___ b. Complete the Response definition with the following:

STATUS CODE: 200**SCHEMA:** object**DESCRIPTION:** OK

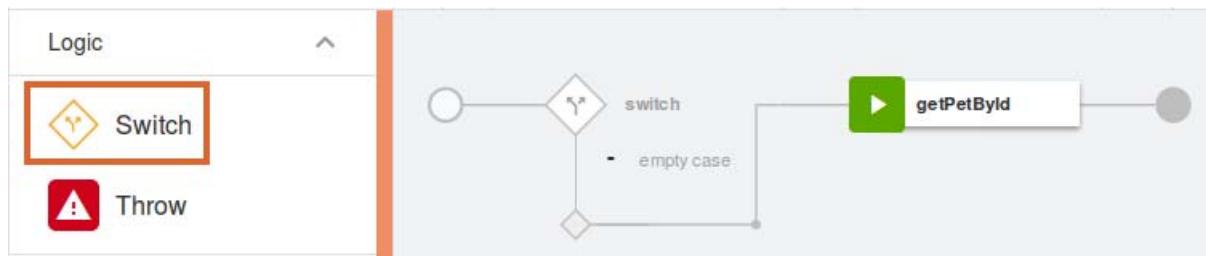
STATUS CODE	SCHEMA	DESCRIPTION	DELETE
200	object	OK	

- ___ c. Click **Save**.
- ___ 6. Create a switch policy to handle two types of API operation requests: GET /pet/petId and POST /pet.

- ___ a. Select the **Assemble** view.

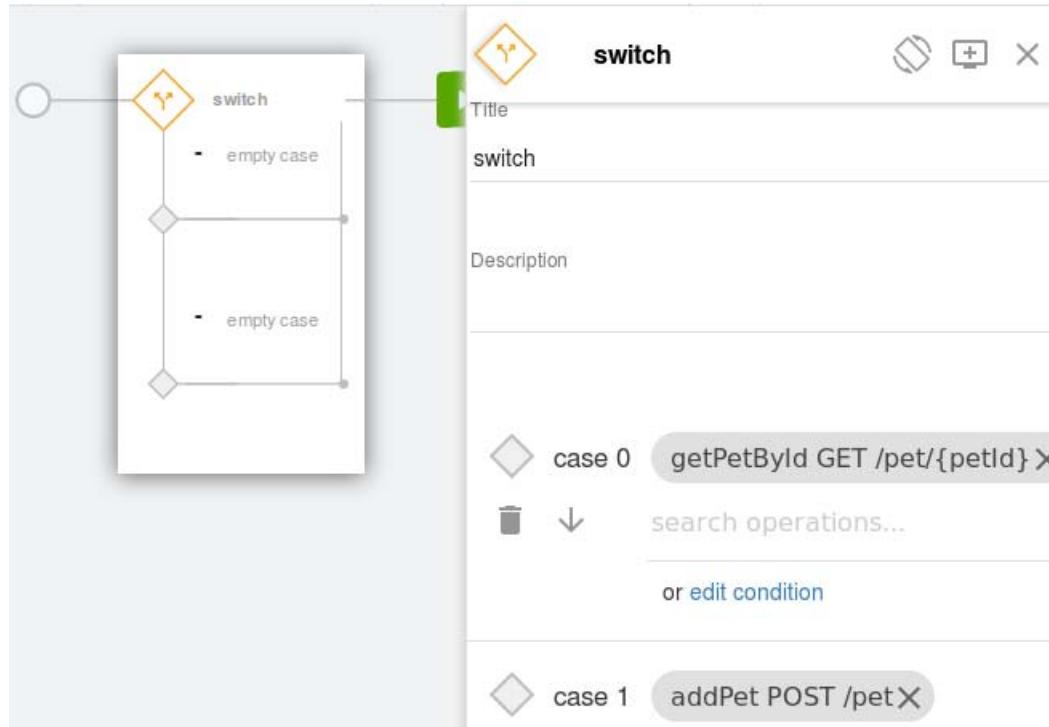


- ___ b. Add a switch policy at the beginning of the message processing pipeline by selecting the Switch from the palette, then drop it between the start and invoke icons on the free-form area.

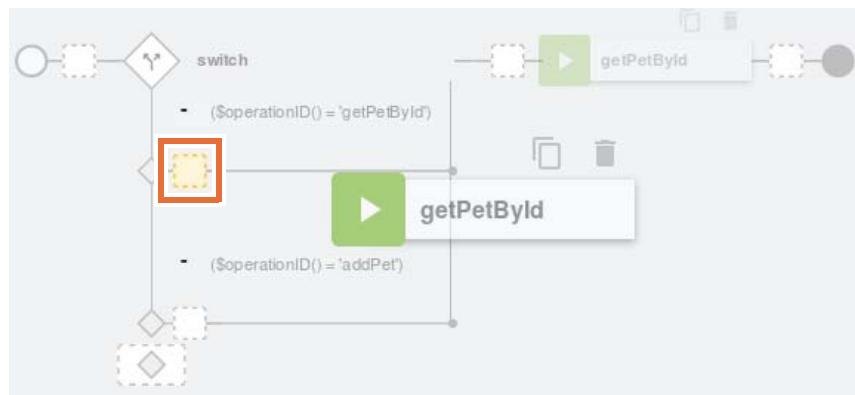


- ___ c. In the properties editor for the switch policy, select the getPetById GET /pet/{petId} for the case 0 branch.
- ___ d. Click + Case to add another case.

- __ e. Set the case 1 branch to addPet POST /pet.

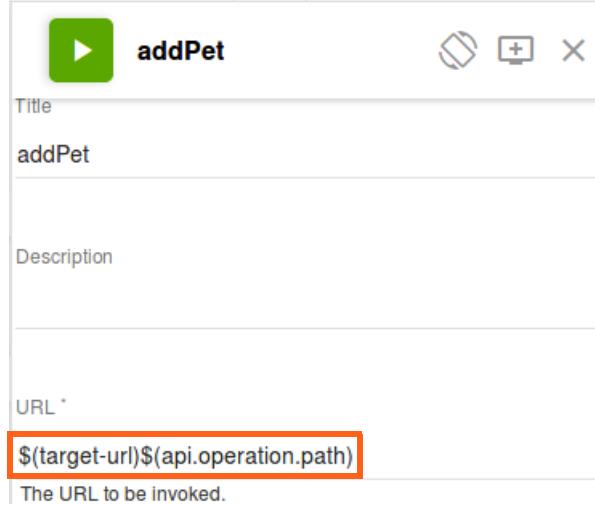


- __ f. Save and close the operation-switch properties editor.
7. Add the invoke policy to the **getPetById** case.
- __ a. Select the existing **getPetById** invoke policy in the assembly palette and drag the policy over the **getPetById** case.

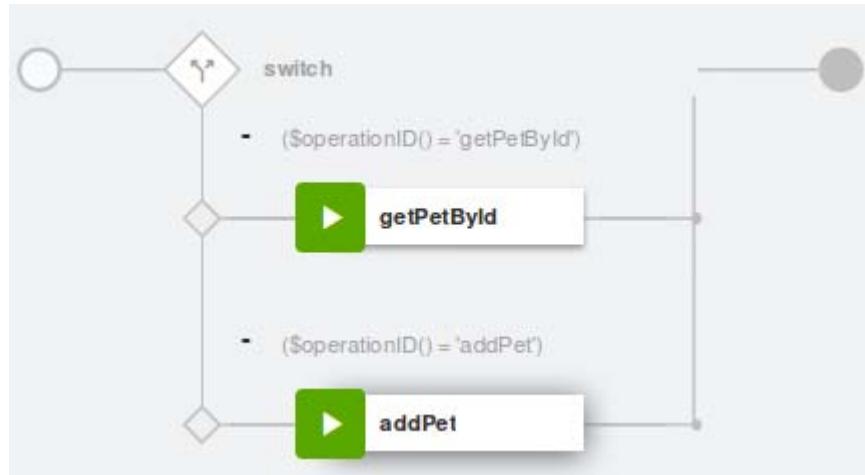


8. Add an invoke policy in the **addPet** case.
- __ a. From the **Policies** palette on the left, select the **invoke** policy.
- __ b. Drag the **Invoke** policy icon into the **addPet** case. This case is the second entry in the switch construct.
- __ c. In the properties editor, change the title to **addPet**.

- ___ d. Set the URL to \$(target-url)\$(api.operation.path)



- ___ e. Scroll down to the **HTTP Method**.
___ f. Change the **HTTP Method** from KEEP to POST.
___ g. Close the assembly properties.
___ h. Verify your API assembly.

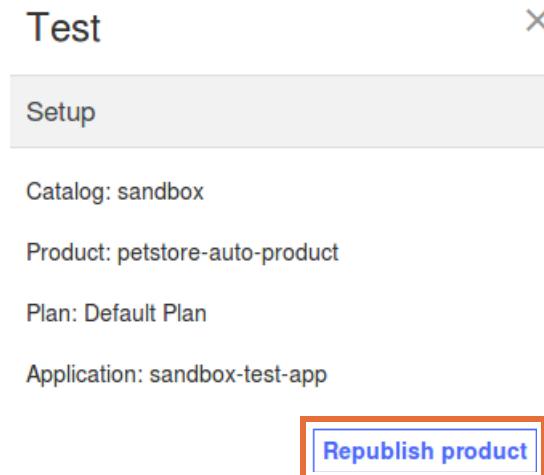


- ___ i. Save the API.

2.6. Test the petstore API with the assembly test feature

In this section, test the POST operation from the built-in test client by adding a pet to the pet store.

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



- ___ c. Select the **post /pet** operation.
- ___ d. Scroll to the Operation section and click **Generate**.

The sample values that you typed when setting the plan definition are inserted in JSON format.
- ___ e. Replace the **id** and **name** values with unique values to use for your testing. Enter a new unique value for a new pet.

__ f. Remove the quotation marks that surround the numeric values.

The screenshot shows a user interface for invoking a REST API. At the top, it says "Choose an operation to invoke:" followed by a dropdown menu set to "post /pet". Below this is a section titled "parameters" containing a JSON object. The JSON object has a required parameter "pet" which is itself another object. This inner object contains an "id" field with the value "98712", which is highlighted with a red rectangular box. The JSON structure is as follows:

```
pet object
pet *
{
  "id" 98712
  "name": "doggie98712"
}
```

At the bottom of the interface, there are two buttons: "Show schema" and "Generate".

__ g. Click **Invoke**.

- ___ h. Confirm that the API implementation returns a status of 200 OK, with the pet information you entered.

Response

Status code:
200 OK

Response time:
176ms

Headers:

apim-debug-trans-id: 426530149-Landlord-
apiconnect-1607dc1d-44a0-4ac4-
bef4-65556c194a94
content-type: application/json
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,96;

Body:

```
{  
  "id": 98712,  
  "name": "doggie98712",  
  "photoUrls": [],  
  "tags": []  
}
```

- ___ 2. To verify the addition of the pet to the Petstore database, invoke the GET operation by using the id value.
- ___ a. Scroll to the Operation section and select **get /pet{petId}**.

- ___ b. Enter the **id** value that you used when performing the post operation to add the pet.

Operation

Choose an operation to invoke:

Operation

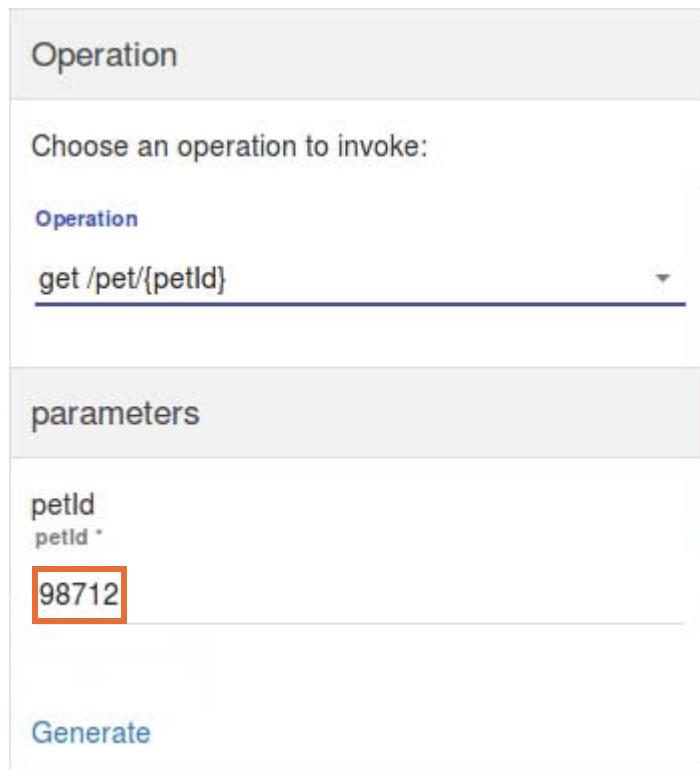
get /pet/{petId}

parameters

petId
petId *

98712

Generate



- ___ c. Click **Invoke**.
___ d. Verify that the results are the same as the POST operation.

The pet object is returned with the **id** and **name** values you provided in the addPet operation.

- ___ e. Close the browser.
___ f. Close the terminal.

End of exercise

Exercise review and wrap-up

In the first part of the exercise, you reviewed the operations for the Swagger Petstore sample REST API application. You then created an OpenAPI definition in the API Manager web application from an existing API. You defined the API paths, operations, request, and response message that describe the petstore API.

In the second part of the exercise, you defined a second API operation for the petstore 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. Defining an API that calls an existing SOAP service

Estimated time

00:30

Overview

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. 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

With API Connect, you can define an API from existing enterprise services. 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.

In this scenario, your organization already developed a set of SOAP web services. The goal of this exercise is to take the operations from the existing SOAP service and make it available at the API Gateway.

To accomplish this, you build a SOAP web service proxy for an existing service. This configuration does not convert 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

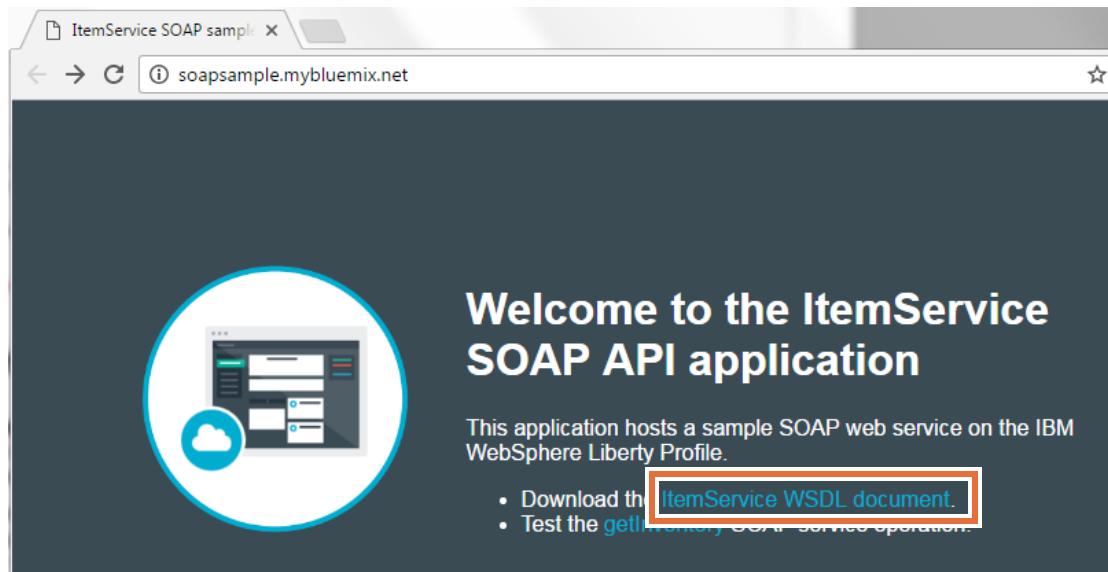
You must complete this lab exercise in the Ubuntu host environment.

3.1. Review the existing SOAP web service

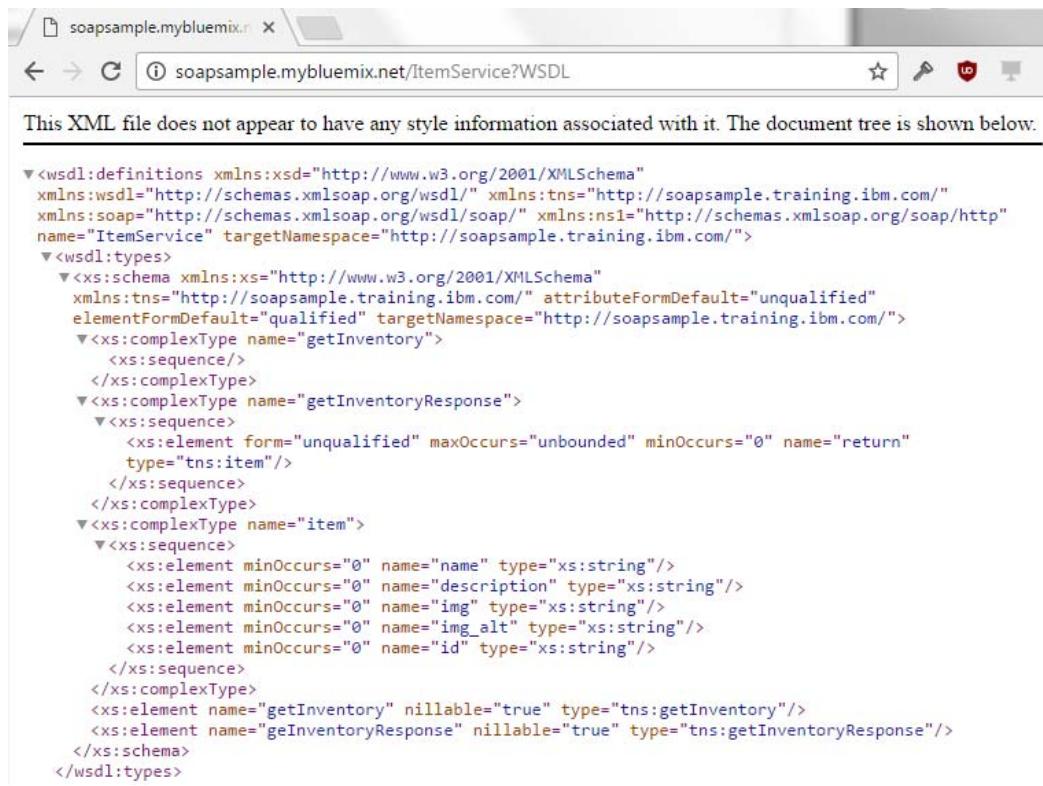
The `ItemService` application maintains a list of vintage IBM products from its corporate history. The application hosts a remote service that returns a list of items in the store inventory. Unlike many REST services on API Connect, the `ItemService` remote service is built as a SOAP web service.

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 browser.
 - ___ 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>

```



Questions

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 binding sections explain 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 by clicking Back on the browser.
 - ___ b. Click **getInventory** to test the SOAP service operation.
 - ___ c. Review the SOAP request and response messages.

ItemService SOAP API Sample

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

SOAP request:

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

SOAP response:

HTTP status code 200

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope
/"><soap:Body><ns2:geInventoryResponse
xmlns:ns2="http://soapsample.training.ibm.com/"><return><ns2:name>Dayton Meat
Chopper</ns2:name><ns2:description>Punched-card tabulating machines and time
clocks were not the only products offered by the young IBM. Seen here in 1930,
manufacturing employees of IBM's Dayton Scale Company are assembling Dayton
Safety Electric Meat Choppers.</ns2:description><ns2:img>images/items/meat-
chopper.jpg</ns2:img><ns2:img_alt>Meat Chopper</ns2:img_alt><ns2:id>1</ns2:id>
</return><return><ns2:name>Hollerith Tabulator and Sorter</ns2:name>
<ns2:description>This equipment is representative of the tabulating system
invented and built for the U.S. Census Bureau by Herman Hollerith (1860-1929).
After observing a train conductor punching railroad tickets to identify
passengers, Hollerith conceived and developed the idea of using punched holes to
record facts about people. These machines were first used in compiling the 1890
Census.</ns2:description><ns2:img>images/items/hollerith-tabulator.jpg</ns2:img>
<ns2:img_alt>Tabulator</ns2:img_alt><ns2:id>2</ns2:id></return><return>
<ns2:name>IBM 77 Electric Punched Card Collator</ns2:name><ns2:description>The
IBM 77 electric punched card collator performed many card filing and pulling
operations. Introduced in 1937, the IBM 77 collator rented for $80 a month. It
was capable of handling 240 cards a minute, and was 40.5 inches long and 51
inches high.</ns2:description><ns2:img>images/items/electric-card-
```



Information

The ItemService website includes a test client for the SOAP service. This page sends a 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...**.

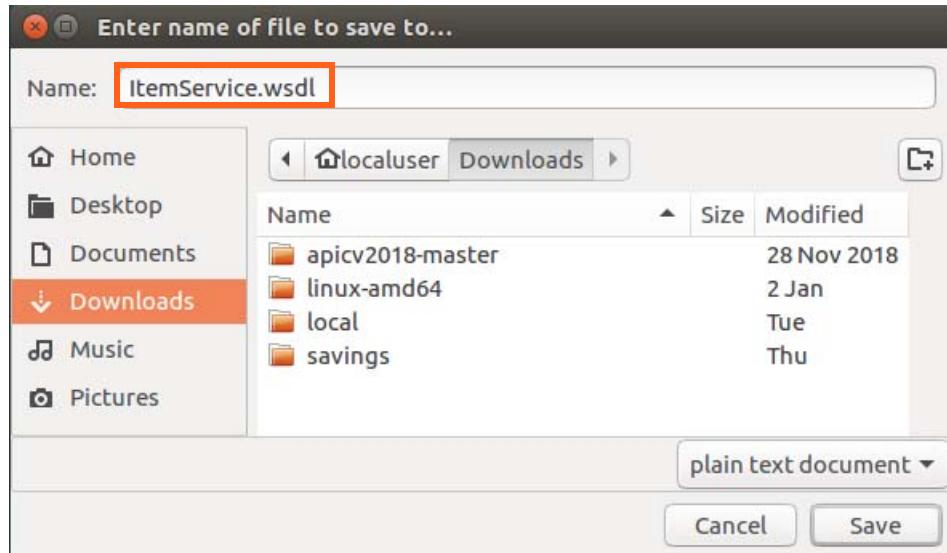
Welcome to the ItemService SOAP API application.

This application hosts a sample SOAP web service on the IBM WebSphere Liberty Profile.

- Download the [ItemService WSDL document](#)
- Test the [getInventory](#) SOAP se

Open Link in New Tab
Open Link in New Window
Open Link in New Private Window
Bookmark This Link
Save Link As...

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



Note

Your list of files might be different than the screen capture.

- __ e. Click **Save**.

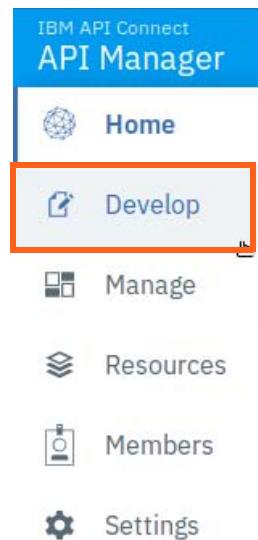
By default, the file is saved to the `/home/localuser/Downloads` directory.

- __ f. Close the web browser with the sample SOAP application.

3.2. Create a SOAP API definition from a WSDL document

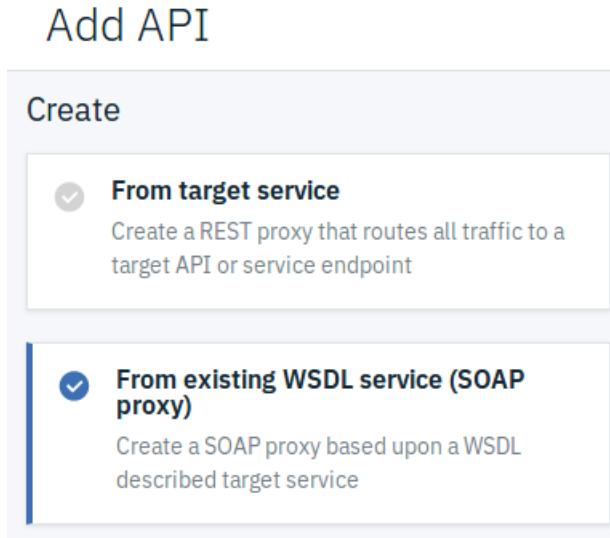
In this section, you generate an OpenAPI 2.0 API definition from an existing SOAP service. Import the WSDL document into the API Manager. Examine the API definition and message processing policies for the API.

- ___ 1. Sign in to API Manager.
 - ___ a. Open a browser window.
Then, type `https://manager.think.ibm.`
 - ___ b. Type the credentials:
 - Username: ThinkOwner
 - Password: Passw0rd!
 Click **Sign in**.
You are signed in to API Manager.
- ___ 2. Select the Develop option from the side menu.



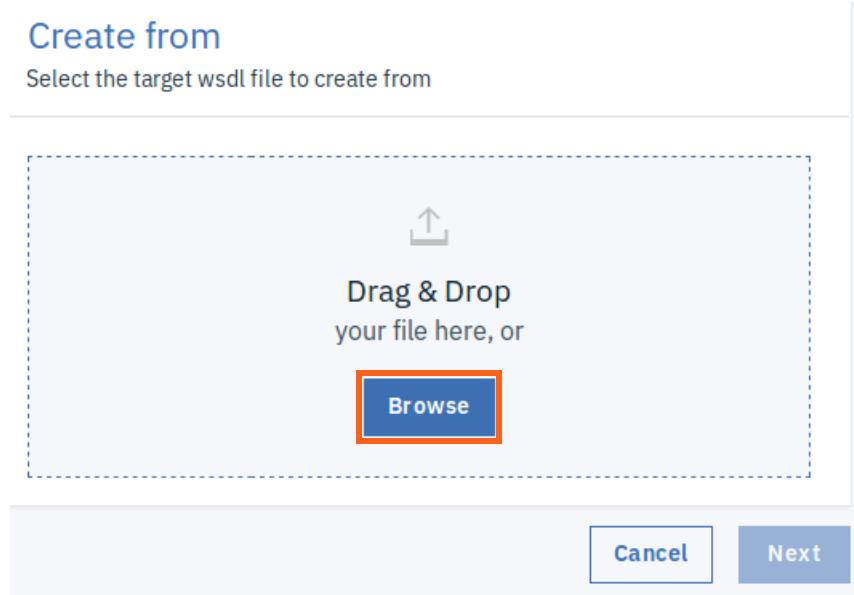
- ___ 3. Create an API definition that is named `ItemService` from the `ItemService` WSDL document.
 - ___ a. In the API Manager develop page, click **ADD** and select **API**.

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



The selector is highlighted.

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



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

- __ g. The ItemService from the imported WSDL is selected. Click **Next**.

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

Back **Cancel** **Next**



Information

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



Troubleshooting

If you see a `formData` is undefined error, you can continue the exercise. This error does not impact subsequent steps.

X API Connect

https://manager.think.ibm.com/manager/think/develop/add-api/ ... ☰

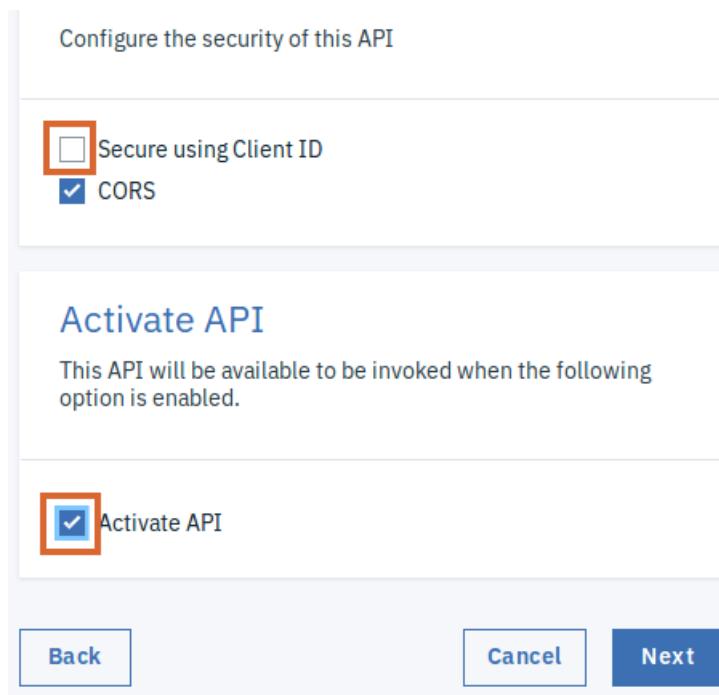
Getting Started Cloud Manager API Manager API Developer Portal DataPower API Gate...

Create API from Existing WSDL Service (SOAP)

Error
formData is undefined

- __ h. Click **Next** on the Info page.
- __ i. Enter the following on the **Configure the security of this API** page:
- Clear the option to **Secure using Client ID** and make sure to select the **CORS** option.

- __ j. Select the **Activate API** option .



- __ k. Click **Next**.



Information

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.

- __ I. The Summary window is displayed.

The screenshot shows the 'Summary' section of an API configuration interface. It displays two green checkmarks: 'Generated OpenAPI 2.0 definition' and '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 string of characters). A large blue 'Edit API' button is located at the bottom right of the summary area.

- __ 4. Review the API definition

- __ a. Click **Edit API**.

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

The screenshot shows the 'ItemService' API setup interface. The left sidebar has 'API Setup' selected. The main area displays basic API 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 'ItemService' API setup interface. The left sidebar has 'API Setup' selected. The main area displays the base path configuration:

- Base Path:** 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.

- ___ 5. Review the message processing policies in the ItemService API definition.
 - ___ a. Select the **Paths** section.
A single path that is named `/getInventory` is displayed.
 - ___ b. Click the ellipsis in the paths area and click **Edit**.

NAME
<code>/getInventory</code>

Add

Edit **Delete**

The Itemservice API defines one POST API operation.

- ___ c. Click the option to edit the POST operation.

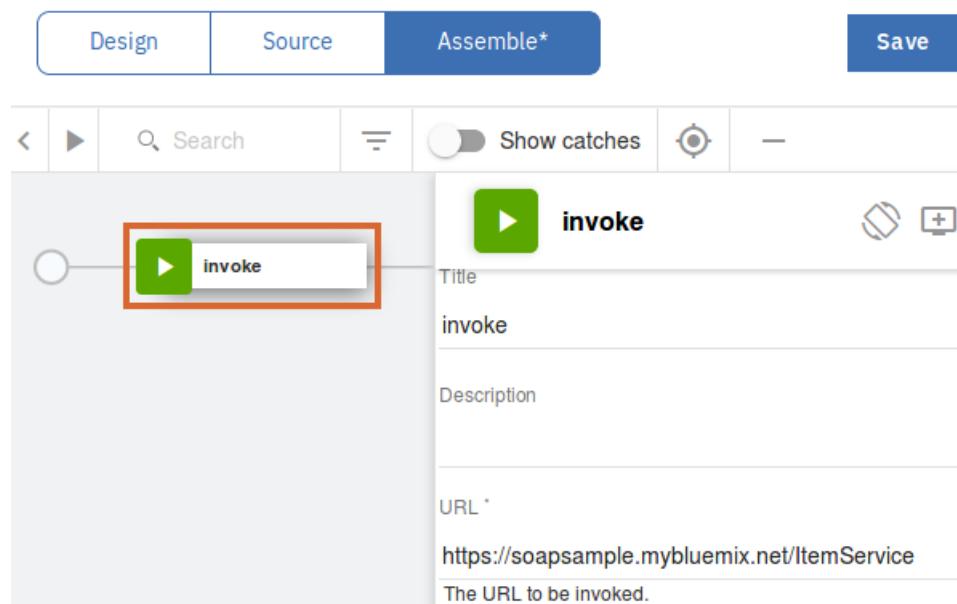


Information

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.
- ___ 6. Review the API definitions.
 - ___ a. Click **Definitions**.
A number of schema definition objects are already defined.
- ___ 7. Examine the message processing policies for the Itemservice API definition.
 - ___ a. Click the **Assemble** tab.

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



Information

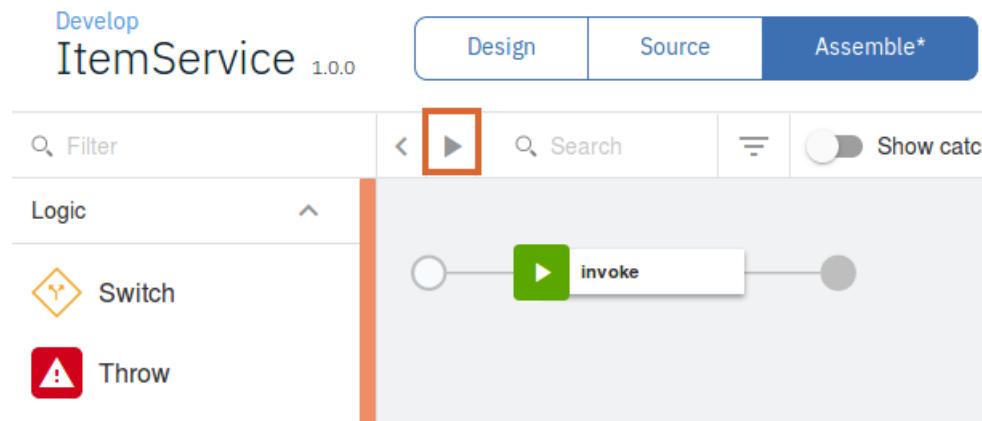
The assemble view displays the message processing policies for all operations in the `Itemservice` API. At run time, the API Gateway enforces these policies on every request message that clients send to the API.

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

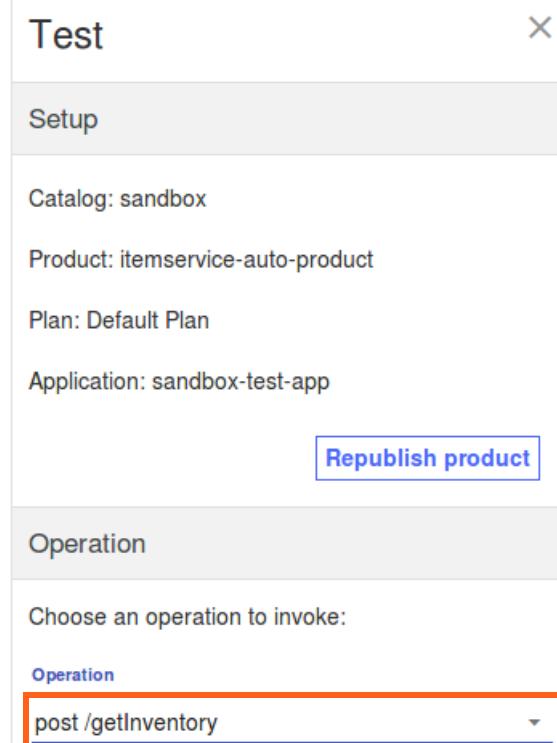
In this section, you test an API that contains DataPower policies on the DataPower Gateway. You use the test feature of the assembly to test the API.

- ___ 1. Test the `Itemservice` API from the test option in the assembly of the API Manager.
 - ___ a. In the Assembly editor, click the **Test** icon.



Since you selected the **Make API active** option earlier, the product is already published to the sandbox catalog and the gateway.

- ___ b. Select the `post/getInventory` operation in the test client.



A client ID is automatically inserted into the test client.

- ___ c. Scroll down to the body area in the test client.

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

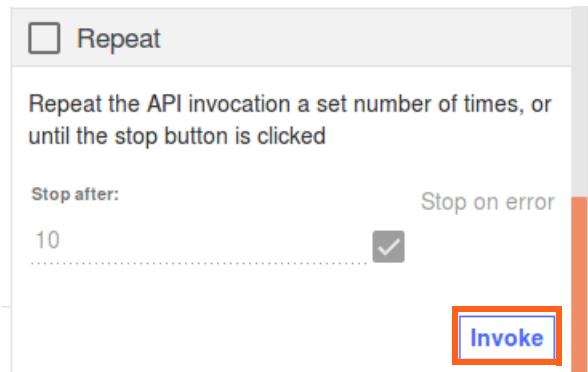


- ___ e. The body area is populated with data.

The screenshot shows a test client window titled 'Test'. Inside, there's a 'parameters' section with a 'body' field containing 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>
```

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



The result is displayed in the test window.

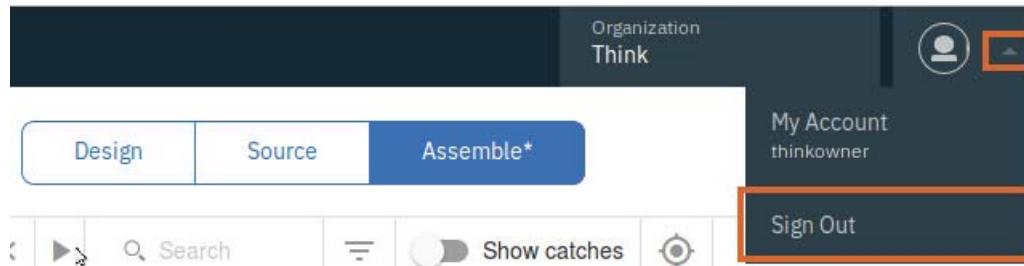
The screenshot shows a 'Test' window with the following details:

- 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>
```

- ___ g. Close the test window.

- __ 2. Sign out of API Manager.



- __ 3. Close the browser.
-



Troubleshooting

If you receive a **401 Unauthorized** error with the message Client id not registered

Response

Status code:
401 Unauthorized

Response time:
36ms

Headers:
apim-debug-trans-id: 426530149-Landlord-
apiconnect-5ef29733-b9b1-4b8e-
b767-65556c19f9ba
content-type: application/json

Body:

```
{
  "httpCode": "401",
  "httpMessage": "Unauthorized",
  "moreInformation": "Client id not registered."
}
```

This occurs when the ItemService API has been published but the two virtual machines got out of sync. To resolve this issue, republish the ItemService API to reregister the client id, then generate a new SOAP envelope.

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

The LoopBack framework is a framework to build REST APIs in the Node.js programming language. This open source framework provides nearly code-less 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, applications 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. You also identify the structure of a LoopBack application and build model objects with the command line utility. After that, you review and test the REST APIs with the Loopback API explorer.

Requirements

You must complete this lab exercise in the Ubuntu host environment.

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. In this section, you generate a sample LoopBack application with the `apic` CLI.



Attention

Do not perform the steps on this page. This is for your reading purposes only.

The `apic lb` command does not work and will corrupt the lab in this VM due to the system date being set to January 25, 2020. Note in an environment with the correct system date, these steps can be followed to install the LoopBack application. That is the reason these steps are being included here. Review the steps in this section (4.1) but do not perform them, then proceed to section 4.2 and complete the lab.

- 1. Create a directory to hold the contents of the LoopBack application.

Do not perform this step. This is for your reading purposes only.

- a. Open a terminal window from the Ubuntu desktop launcher.
- b. Create two directories, `samples` and `samples/notes`.

```
$ mkdir samples samples/notes
$ cd samples/notes
```
- c. Verify that your current working directory is the `sample/notes` directory.

```
$ pwd
```

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

Do not perform this step. This is for your reading purposes only.

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

```
$ apic lb
```
- b. Type `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. In this section, you review the API definition files, the configuration settings, the model properties, and the model source code.

- ___ 1. Review the structure of the application.
 - ___ a. If you don't have a terminal window open, open a terminal window from the Ubuntu desktop launcher.
 - ___ b. Access the samples/notes directory by entering the command:
`$ cd samples/notes`
 - ___ c. List the contents of the root directory in the 'notes' application.
`$ ls`
 localuser@ubuntu:~/samples/notes\$ ls
 client node_modules package.json server
 common notes.yaml package-lock.json



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.

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

```
$ ls common
models
$ ls 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.

- ___ e. 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.

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

```
$ ls 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"
    }
  }
}
```

- ___ 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.

- __ 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 the 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. 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.

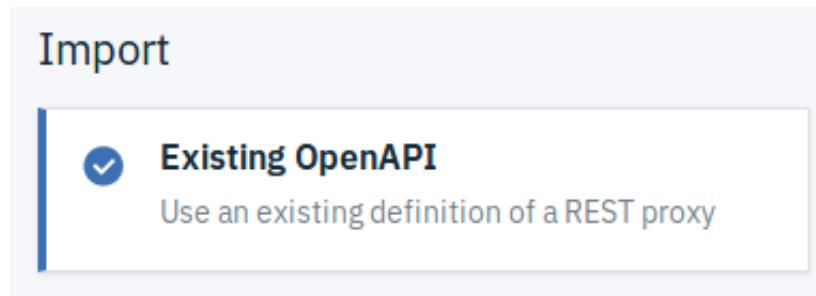
___ c. Verify that the file is there by entering the `ls` command to view the directory.

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.

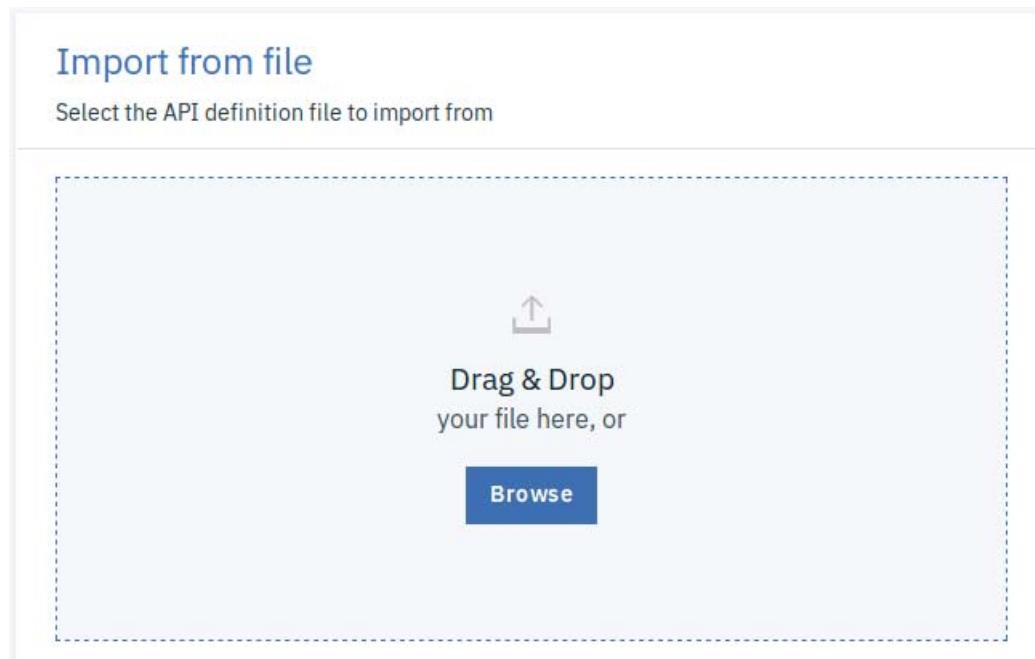
- ___ 1. Start the API Manager web application.
 - ___ a. Sign on to the API Manager web interface from a browser session. Type the address <https://manager.think.ibm>
 - ___ b. Sign in with the username and password that is associated with your API Manager account:
 - Username: ThinkOwner
 - Password: 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**.



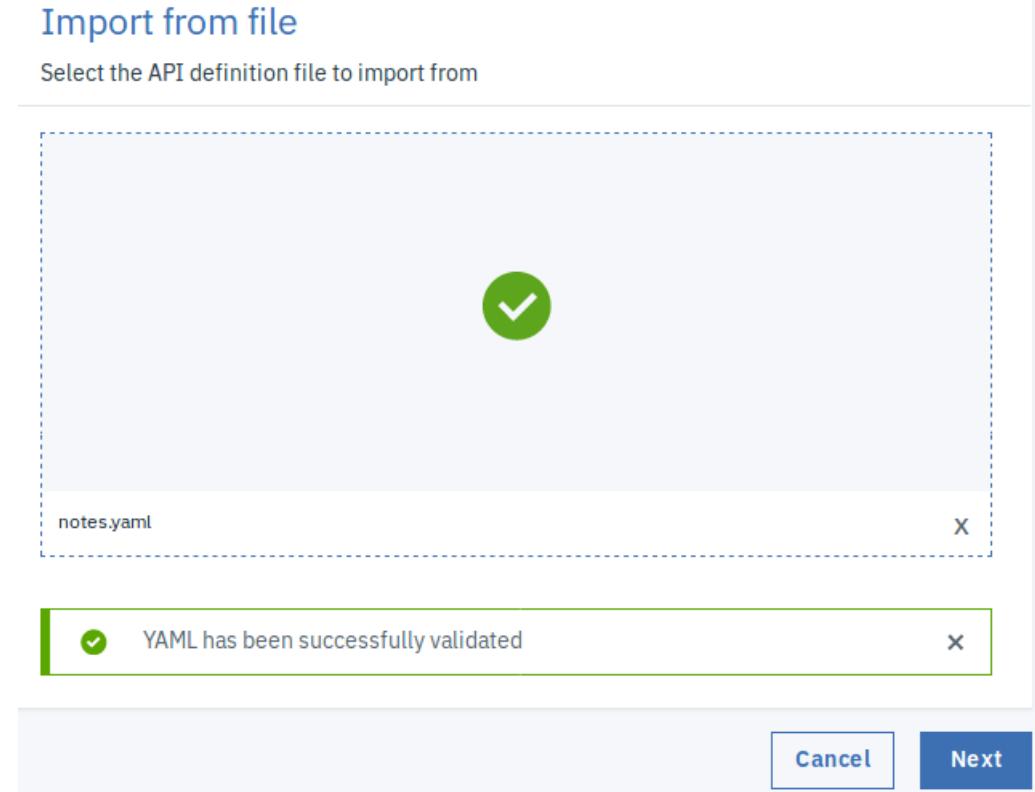
The selector is displayed.

- ___ d. Click **Next**.

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

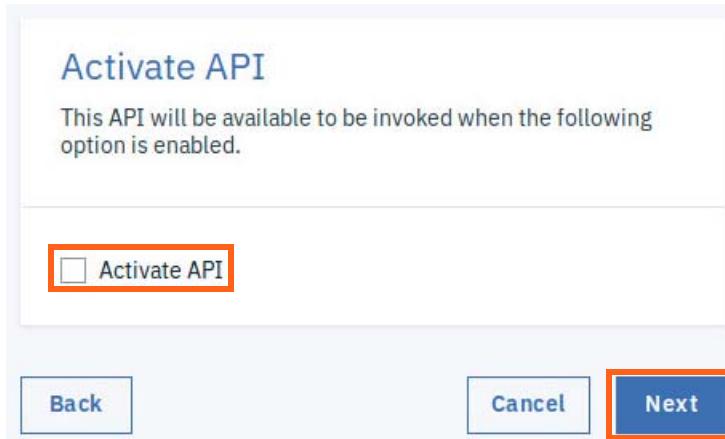


- __ f. Navigate to the ~/home/localuser/samples/notes directory, then, select the notes.yaml file. Click **Open**.
__ g. The YAML file is imported and successfully validated.

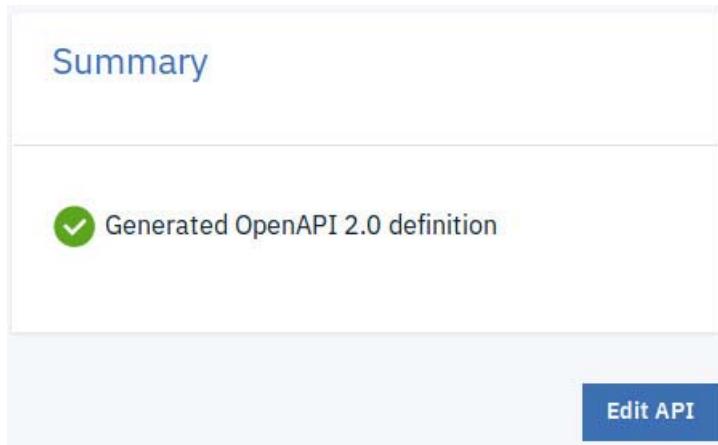


- __ h. Click **Next**.

- i. Since you are going to test the application with the Loopback API Explorer, leave the Activate API option cleared and click **Next**.



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

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.



Information

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 title, name, version, and description of the API.

A screenshot of the API Editor interface. At the top, there's a header with tabs: 'Develop' (highlighted in blue), 'notes' (highlighted in orange), and '1.0.0'. Below the tabs are three main sections: 'API Setup' (highlighted in blue), 'Info', and 'Definitions'. The 'API Setup' sidebar contains links for 'Security Definitions', 'Paths', 'Definitions', 'Properties', 'Target Services', 'Categories', and 'Activity Log'. The 'Info' section has fields for 'Title' (set to 'notes'), 'Name' (set to 'notes'), and 'Version' (set to '1.0.0'). The 'Definitions' section is currently empty.

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

The screenshot shows the 'API Setup' interface with a sidebar on the left containing links: Security Definitions, Security Paths, Definitions, Properties, Target Services, Categories, and Activity Log. The 'Host' section is expanded, showing the definition 'The host used to invoke the API' and an optional input field. The 'Base Path' section is also expanded, showing the definition 'The base path is the initial URL segment of the API; paths or operations' and an optional input field containing '/api'.

API Setup	
Host	The host used to invoke the API
Address (optional)	
Base Path	The base path is the initial URL segment of the API ; paths or operations
Base path (optional)	/api



Information

The **host** property stores the hostname 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 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 is a sidebar with the following options:

- API Setup (selected)
- Security Definitions
- Security Paths
- Definitions
- Properties
- Target Services
- Categories
- Activity Log

The main area is divided into two sections: 'Consumes' and 'Produces'.

Consumes

- Consumes (optional)**
 - application/json
 - application/xml
- Add media type (optional)**
text/xml

Produces

- Produces (optional)**
 - application/json
 - application/xml
- Add media type (optional)**
text/javascript



Information

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** sections set 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 and click **Edit**.

Paths	
API Setup Security Definitions Security Paths Definitions Properties Target Services	<div style="border: 1px solid #ccc; padding: 10px;"> <p>NAME</p> <p>/Notes</p> <p>/Notes/replaceOrCreate</p> <p>/Notes/upsertWithWhere</p> <p>/Notes/{id}/exists</p> <p>/Notes/{id}</p> </div> <div style="position: absolute; right: -10px; top: 0; background-color: white; border: 1px solid #ccc; padding: 5px; font-size: small;"> Edit Delete </div>

The list of operations is displayed.

The screenshot shows the LoopBack API definition interface. At the top, there is a 'Path name' input field containing '/Notes'. Below it, a 'Path Parameters' section has an 'Add' button. A table for 'Operations' lists five rows: POST, PATCH, PUT, and GET, each with an ellipsis button ('...').

REQUIRED	NAME	LOCATED_IN	TYPE	DESCRIPTION	DELETE
	POST				...
	PATCH				...
	PUT				...
	GET				...



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**.

-
- ___ e. Click the return arrow or click **Cancel** to go back to the API definition.
 - ___ f. Click **Definitions**
 - ___ g. Select the ellipsis in the /Notes path. 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>.

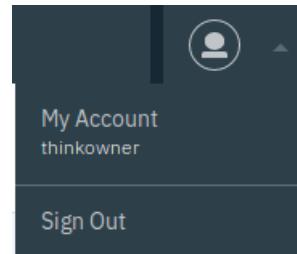
A one-to-one correlation exists between the structure of your LoopBack model objects and the data type definitions in this section of the API definitions file.

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 section, you test the API operations with the test feature of LoopBack, the API Explorer.

- 1. Start the notes Node application.

- a. In the terminal, from the /home/localuser/samples/notes directory, type:

```
npm start
```

- b. The output is displayed:

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

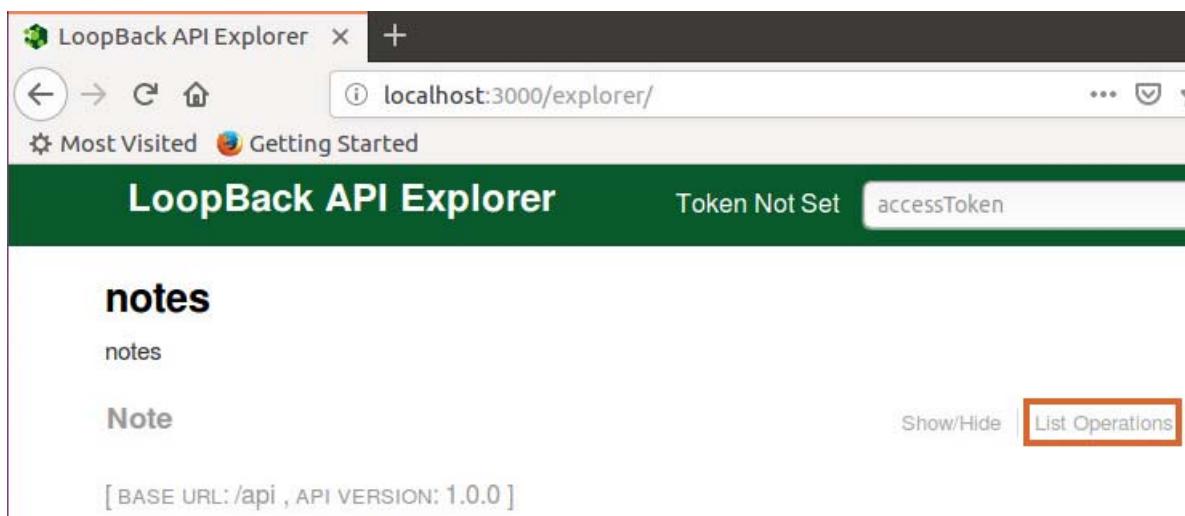
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- 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, there's a green header bar with the title "LoopBack API Explorer", a "Token Not Set" status message, and a text input field labeled "accessToken". To the right of the input field is a green button labeled "Set Access Token". Below the header, the word "notes" is displayed in large bold letters. Underneath, the word "notes" appears again in a smaller font. A section titled "Note" is shown with four operations listed:

- 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.

The "POST /Notes" operation is highlighted with a red border around its row.

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

- __ b. Click inside the example value area in the Parameters area on the page.

The screenshot shows the LoopBack API documentation for the 'Line' endpoint. The URL is `/Notes`. The 'Method' is `POST` and the 'Action' is `Line`. The description is "Create a new instance of the model and persist it into the data source".

Response Class (Status 200)
Request was successful

Model Example Value

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

Response Content Type `application/json`

Parameters

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

Parameter content type:
`application/json`

The example data is copied to the data area.

- __ c. Change the values in the data area.

```
"title": "hello"
"content": "world"
```

The screenshot shows the same LoopBack API documentation page after changing the values in the 'data' parameter. The 'Example Value' field now contains the modified JSON object:

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

- __ d. Click **Try it out**.

- ___ e. 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**.

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

The screenshot shows the API Explorer interface with the following sections:

- Try it out!**: A button to execute the curl command.
- Hide Response**: A link to collapse the response section.
- Curl**: The curl command used: `curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/Notes'`.
- Request URL**: The URL entered: `http://localhost:3000/api/Notes`.
- Response Body**: The JSON response received:

```
[{"title": "hello", "content": "world", "id": 1}]
```
- Response Code**: The status 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

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 Ubuntu host environment. This exercise must be completed to complete the remaining exercises in this course.

5.1. Create the inventory application

In this section, you 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.



Attention

Do not perform the steps in this section.

The `apic lb` command does not work and will corrupt the lab in this VM due to the system date being set to January 25, 2020. Note in an environment with the correct system date, these steps can be followed to install the LoopBack application. That is the reason these steps are being included here. Review the steps in this section (5.1) but do not perform them, then proceed to section 5.2 and complete the lab.

- ___ 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 (`/home/localuser/`) create a directory that is named `inventory`.

```
$ cd ~
$ mkdir ~/inventory
$ cd 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.
- ___ 4. Install the LoopBack MySQL connector into the `inventory` application by running the command:

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

5.2. Create the MySQL data source

- ___ 1. In the terminal, change the directory to inventory.

```
$ cd inventory
```

- ___ 2. Examine the dependencies section of the project manifest file by entering the command:

```
$ 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-mongodb": "^3.9.2",
  "loopback-connector-mysql": "^5.3.1",
  "serve-favicon": "^2.0.1",
  "strong-error-handler": "^3.0.0"
},
...
```

- ___ 3. 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`. Enter the **bolded** commands in the following sequence:

```
? 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 (eg:
mysql://user:pass@host/db): <hit Enter>
? host: platform.think.ibm
? port: 3306
? user: localuser
? password: passw0rd
? database: think
```

```
localuser@ubuntu:~/inventory
localuser@ubuntu:~$ cd inventory
localuser@ubuntu:~/inventory$ apic lb datasource
? Enter the datasource name: mysql-connection
? Select the connector for mysql-connection: MySQL (supported by StrongLoop)
? Connection String url to override other settings (eg: mysql://user:pass@host/d
b):
? host: platform.think.ibm
? port: 3306
? user: localuser
? password: *****
? database: think
localuser@ubuntu:~/inventory$
```

- ___ 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

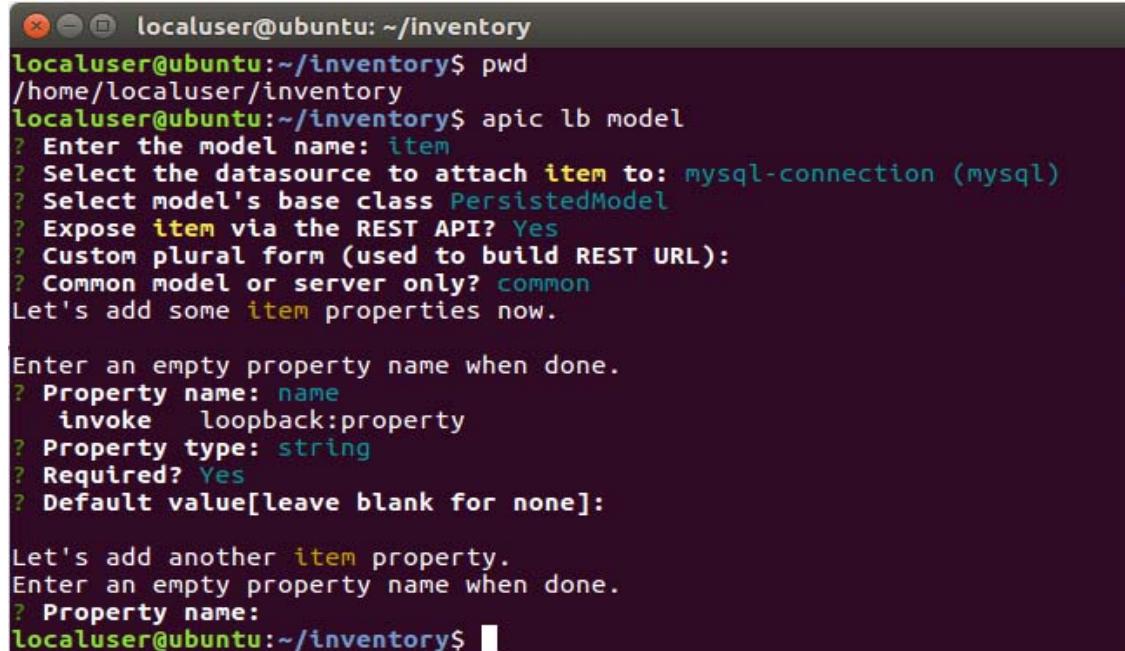
The `apic lb datasource` command adds a data source definition to your LoopBack application. The data source relies on the LoopBack connector: a npm package that implements the data source object. You can modify the settings directly in the `server/datasources.json` configuration file.

5.3. 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, you define the properties in the item model and then define the model properties. Then, you 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`
 - c. Enter the following properties for the `item` model. Enter the **bolded** commands in the following sequence:

```
? 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}: (Press Enter)
Enter an empty property name when done.
Press enter to finish.
```



A screenshot of a terminal window titled "localuser@ubuntu: ~/inventory". The terminal shows the process of creating a new LoopBack model named "item". The user runs the command "apic lb model" and is prompted to enter the model name, which is "item". They then select the datasource as "mysql-connection (mysql)" and choose "PersistedModel" as the base class. The model is exposed via the REST API (Yes). The custom plural form is left blank. The model is defined as a common model. The user is then prompted to add properties. They add a property named "name" of type "string", which is required and has no default value. The user is then prompted to add another property, but exits the loop by pressing enter.

```
localuser@ubuntu:~/inventory$ pwd
/home/localuser/inventory
localuser@ubuntu:~/inventory$ 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? Yes
? Custom plural form (used to build REST URL):
? Common model or server only? common
Let's add some item properties now.

Enter an empty property name when done.
? Property name: name
  invoke loopback:property
? Property type: string
? Required? Yes
? Default value[leave blank for none]: 

Let's add another item property.
Enter an empty property name when done.
? Property name:
localuser@ubuntu:~/inventory$
```



Information

You created a model that is named `item` with a single property that is named `name`. Later in this exercise, 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 2. Item model properties

Required	Property name	Type	Description
Yes	name	string	
Yes	description	string	item description
Yes	img	string	location of item image
Yes	img_alt	string	item image title
Yes	price	number	item price
No	rating	number	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
```

```
localuser@ubuntu:~/inventory/common/models
localuser@ubuntu:~/inventory/common/models$ more item.js
'use strict';

module.exports = function(Item) {
};

localuser@ubuntu:~/inventory/common/models$
```

- __ c. Review the item.json file.

```
$ more item.json
```

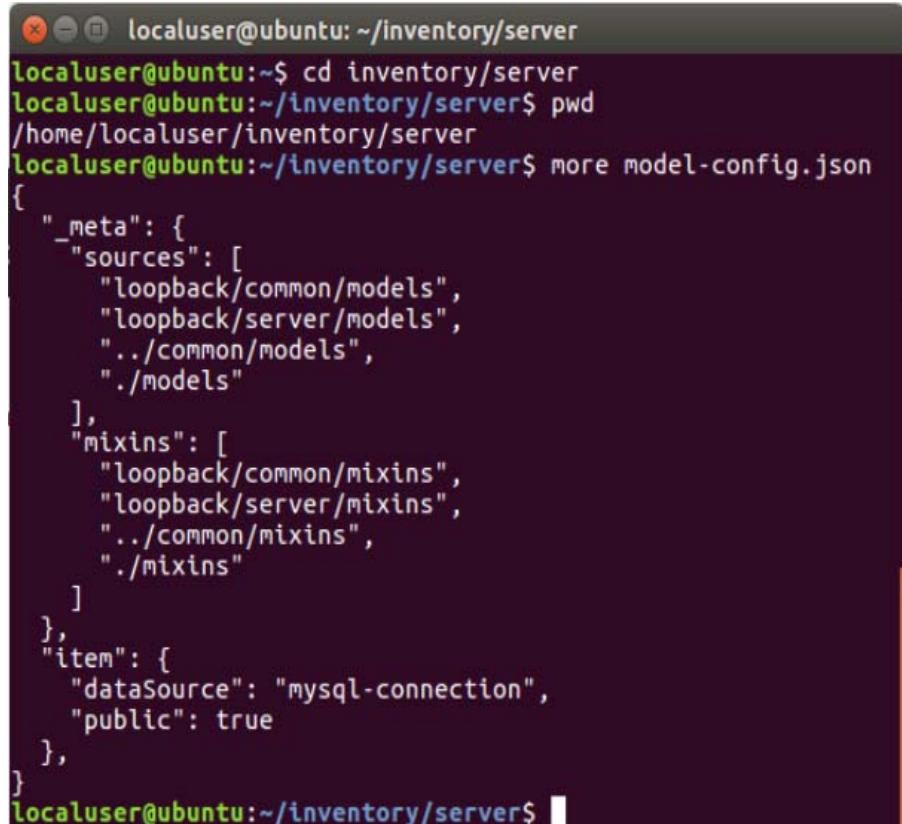
```
localuser@ubuntu:~/inventory/common/models
localuser@ubuntu:~/inventory/common/models$ more item.json
{
  "name": "item",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "name": {
      "type": "string",
      "required": true
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
localuser@ubuntu:~/inventory/common/models$
```

- __ 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
```



```
localuser@ubuntu:~/inventory/server
localuser@ubuntu:~$ cd inventory/server
localuser@ubuntu:~/inventory/server$ pwd
/home/localuser/inventory/server
localuser@ubuntu:~/inventory/server$ 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
  }
}
localuser@ubuntu:~/inventory/server$
```

- ___ 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>

5.4. Test the inventory application and items model

In this section, you start a local copy of the Inventory LoopBack application in the terminal. Then, you examine the API operations for the item model in the LoopBack API Explorer view. Lastly, you 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 'localhost:3000/explorer/#/item'. Below it is a navigation bar with 'Getting Started' and a green header bar titled 'LoopBack API Explorer' with tabs for 'Token Not Set' and 'accessToken'.

The main content area is titled 'inventory'. It shows a list of operations:

- 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.

The screenshot shows the 'item' model in the LoopBack API Explorer. The 'GET /items' operation is highlighted with a red border.

item		Show/Hide List Operations Expand Operations
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.

___ 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 in the late 1800s. It was used to process the 1890 US Census data. The machine used punched cards to store data and could sort, tabulate, and print results. It was the first large-scale mechanical computer and helped revolutionize data processing.",
    "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'
```

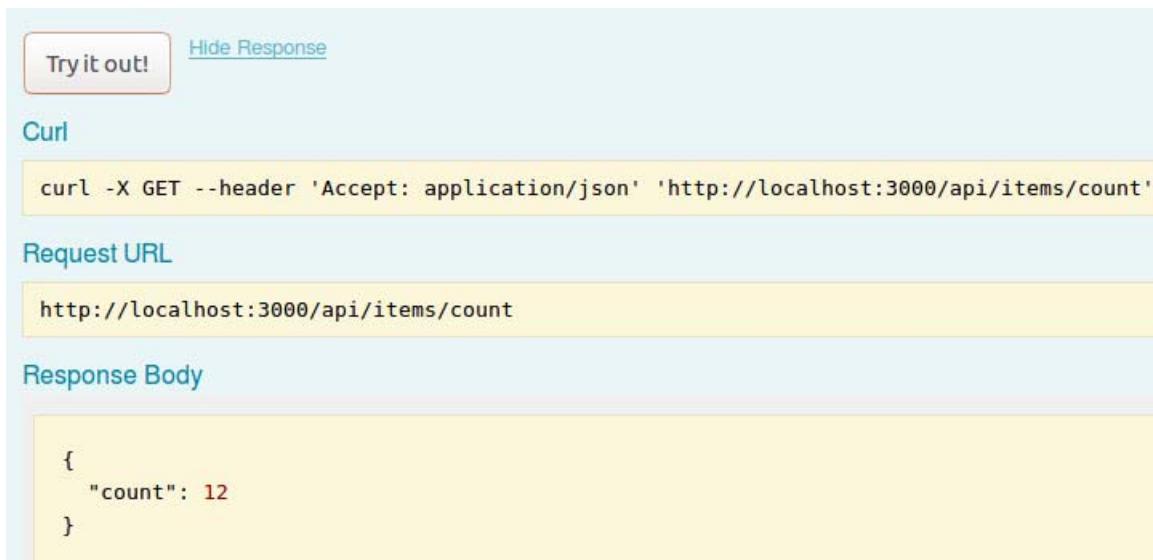
Response

Code: 200

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

```
[
  {
    "name": "Dayton Meat Chopper",
    "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.

5.5. Create a MongoDB data source and the review model

In this section, you define a second model object that is named `ratings` that represents 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 this 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.

- ___ 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 connect

```
? 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): <hit Enter>
? host: platform.think.ibm
? port: 27017
? user:
? password:
? database: think
```

```
localuser@ubuntu:~/inventory
^Clocaluser@ubuntu:~/inventory$ apic lb datasource
? Enter the datasource name: monodbd-connection
? Select the connector for monodbd-connection: MongoDB (supported by StrongLoop)
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database):
? host: platform.think.ibm
? port: 27017
? user:
? password:
? database: think
localuser@ubuntu:~/inventory$
```



Information

When the `apic lb datasource` command defines a data source, it checks whether you have the correct LoopBack connector for the data source.

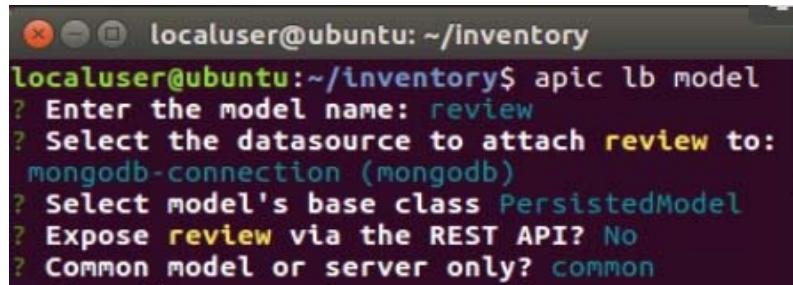
For more information about the MongoDB LoopBack connector configuration and settings, see: <http://loopback.io/doc/en/lb3/MongoDB-connector.html>.

- ___ 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. Enter the following properties for the `review` model. Enter the **bolded** commands in the following sequence:

- ? 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**
- ? Common model or server only?
 > **common**



```
localuser@ubuntu:~/inventory$ apic lb model
? Enter the model name: review
? Select the datasource to attach review to:
mongodb-connection (mongodb)
? Select model's base class PersistedModel
? Expose review via the REST API? No
? 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, entering the bolded commands in sequence.

Enter an empty property name when done.

- ? Property name: **date**
- ? Property type:
 > **date**
- ? Required? **Y**
- ? Default value [leave blank for none]: (**Press Enter**)

d. Enter the remaining properties according to the table.

Table 3. 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 to exit.

Let's add another review property.

Enter an empty property name when done.

? Property name:

```
localuser@ubuntu:~/inventory$
```

5.6. 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. Enter the **bolded** commands in the following sequence:

? 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: (**Press Enter**)

? Require a through model? **No**

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

? Disable the relation from being included? **No**

```
localuser@ubuntu:~/inventory$ apic lb relation
? 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:
? Require a through model? No
? Allow the relation to be nested in REST APIs: No
? Disable the relation from being included: No
localuser@ubuntu:~/inventory$
```

- __ 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 relations definition section in `item.json`.

```
"validations": [],
"relations": {
  "reviews": {
    "type": "hasMany",
    "model": "review",
    "foreignKey": ""
  }
},
"acl": [],
"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. They control how users can create, update, and delete `item` and `review` model objects.

5.7. Test the inventory application and review model

In this section, start a local copy of the Inventory LoopBack application in the terminal. You then examine the API operations for the item model in the LoopBack API Explorer view. Lastly, you 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 .

(node:8679) 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
```

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

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

The LoopBack API Explorer opens in the browser window.

The screenshot shows the LoopBack API Explorer interface. At the top, there's a green header bar with the title "LoopBack API Explorer". To the right of the title, it says "Token Not Set" and has a text input field labeled "accessToken" with a "Set Access" button next to it. Below the header, the word "inventory" is highlighted in blue, indicating the selected model. Under "inventory", there's a sub-section for "item". On the right side of the "item" section, there are three buttons: "Show/Hide" (which is currently active), "List Operations", and "Expand Operations". At the bottom left, there's a note: "[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**.

Earlier you tested the `GET /items` operation that returned data from the MySQL table with the `id` field that has a range 1 - 12.

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.

- ___ 3. 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.

- ___ 4. 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
id	(required)	Item id	path	string
filter			query	string
Try it out!				

Click **Try it out**.

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

The screenshot shows the LoopBack API Explorer interface. At the top, the URL `http://localhost:3000/api/items/1/reviews` is entered. Below it, the **Response Body** section displays the following JSON data:

```
[
  {
    "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
  }
]
```

Below the Response Body, the **Response Code** section shows `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 mongodb database with unstructured data. The MySQL service and the mongodb 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:

```
localuser@ubuntu:~$ sudo service mongod status
[sudo] password for localuser:
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: e
   Active: active (running) since Sat 2020-01-25 05:00:50 EST; 8h ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 8327 (mongod)
    Tasks: 28
   Memory: 186.5M
      CPU: 1min 39.318s
     CGroup: /system.slice/mongod.service
             └─8327 /usr/bin/mongod --config /etc/mongod.conf
```

Jan 25 05:00:50 ubuntu systemd[1]: Started MongoDB Database Server.
lines 1-12/12 (END)

Click CTRL-C to exit

mongo sign on and queries.

```
localuser@ubuntu:~$ mongo
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
think 0.000GB
> use think
switched to db think
> show collections
review
> coll = db.review
think.review
> coll.find();
displays json table
> coll.count();
6
> exit
bye
```

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 the 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 (Exercise 5) in this course. You must complete this lab exercise in the Ubuntu host environment.

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.
 - ____ a. 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"`



```
{
 "restApiRoot": "/inventory",
 "host": "0.0.0.0",
 "port": 3000,
 "remoting": {
 "context": false,
 "rest": {
 "handleErrors": false,
 "normalizeHttpPath": false,
 "xml": false
 }
 }
},
```

- \_\_\_\_ c. Save your changes.

The base path value contains this value when you generate the OpenAPI definition for the inventory application. You also see this value in the 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 when the LoopBack framework creates or updates a persisted model object.

— 1. Open the `common/models/item.js` model script file from the terminal.

- a. Enter `$ cd common/models` to change the directory.
- b. From the terminal, type `gedit item.js`.  
The file opens in the editor.

```
'use strict';

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

- c. 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.

2. 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.

- 
- \_\_\_ 3. Save the `item.js` model script file.

## 6.3. Test the operation hook

In this section, you test the access operation hooks and run the inventory API application from the terminal. You also 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 .


```

```
(node:15690) 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
```

- \_\_\_ 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 **Show/Hide** if the operations are not listed.

- \_\_\_ b. Click the `GET /items/count` in the operations list.

- \_\_\_ c. You see the sample response of the call, and the parameter filter for calling the `GET` operation with curl.

- \_\_\_ d. Click **Try it out**.

- \_\_\_ e. The result is displayed in the API Explorer.

The screenshot shows the API Explorer interface. At the top, there's a "Try it out!" button and a "Hide Response" link. Below that is a "Curl" section containing the command: "curl -X GET --header 'Accept: application/json' 'http://localhost:3000/inventory/items/count'". Under "Request URL", the URL "http://localhost:3000/inventory/items/count" is shown. In the "Response Body" section, the JSON object "{ count: 12 }" is displayed.

- \_\_\_ 3. Confirm that the **access** operation hook logged the query against the **item** database.

- \_\_\_ a. Return to the terminal window where the application was started.

The terminal window shows the command "npm start" being run. The output includes a deprecation warning about the URL string parser, followed by the message "Web server listening at: http://localhost:3000" and "Browse your REST API at http://localhost:3000/explorer". Below this, the log message "Accessed item matching {}" is visible.

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 by right-clicking and selecting:  
**Open a New Document**



— b. Go to the `/home/localuser/lab-files/remote/` folder.

— c. Select the `item.js` script.

Click **Open**.

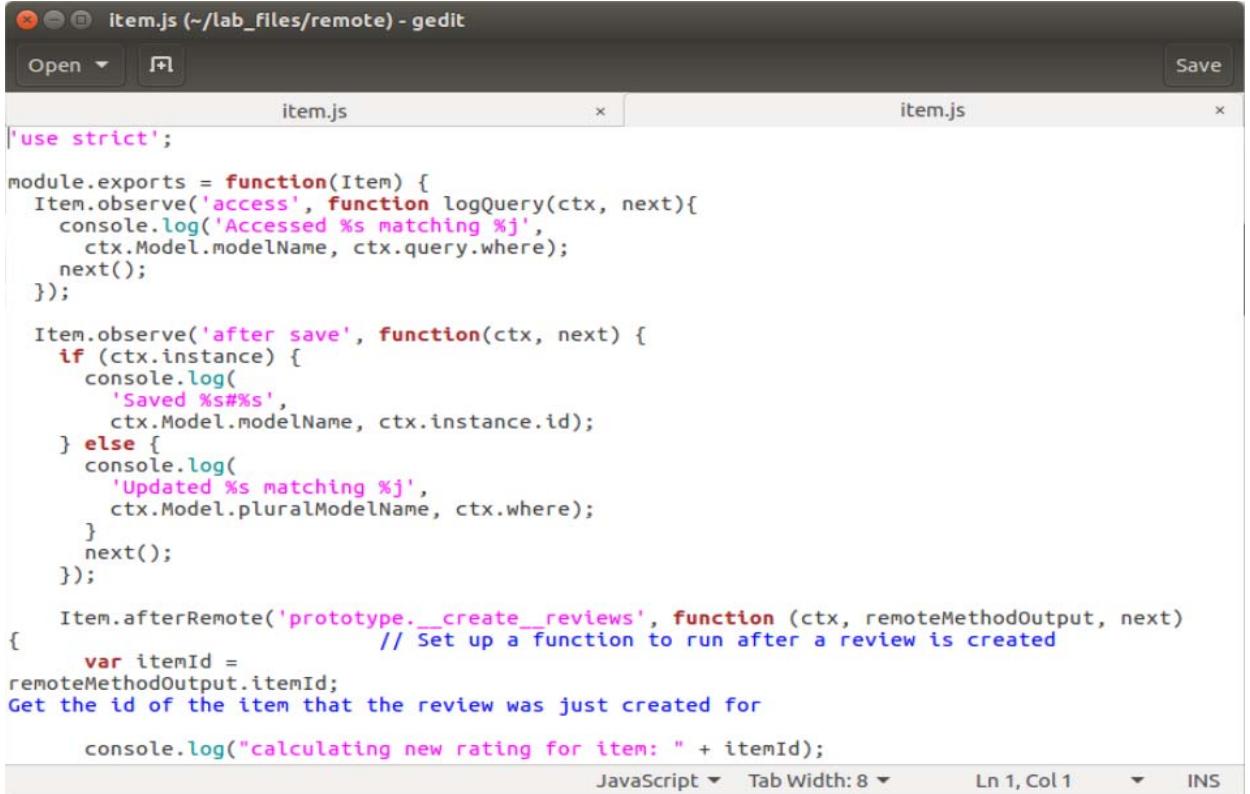
— d. In the document, right-click and select **Select All** to highlight the contents of the `item.js` script.

— 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. Right-click and select **Paste** to paste the contents of the original `item.js` file.

\_\_ h. Click **Save**.



```
'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();
 });

 Item.afterRemote('prototype.__create_reviews', function (ctx, remoteMethodOutput, next)
 {
 var itemId =
 remoteMethodOutput.itemId;
 Get the id of the item that the review was just created for

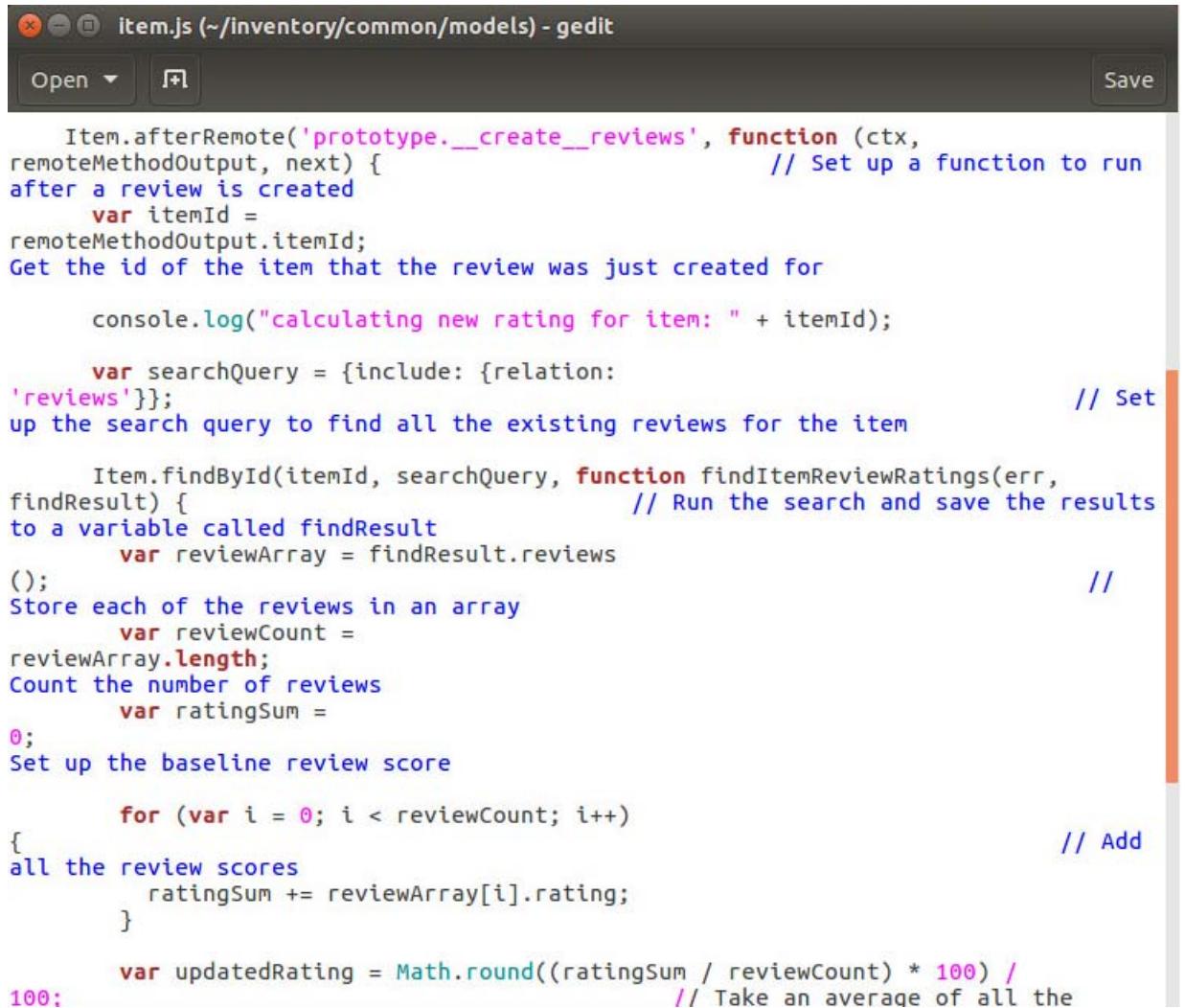
 console.log("calculating new rating for item: " + itemId);
 });
}
```

Save

JavaScript ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

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

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



```

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.

## 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 .


```

```
(node:25053) 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
```

- \_\_\_ 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 Operations**.

- \_\_\_ 2. Create a review for the item record with an identifier value of 1.

- \_\_\_ a. Click **Show/Hide** if the operations are not listed.

- \_\_\_ b. In the API Explorer, select the `post/items/{id}/reviews` operation.

- \_\_\_ c. 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.

- \_\_ d. Click anywhere in the **Example Value** area to copy the sample body content to the **data** area.

| Parameter   | Value | Description | Parameter Type | Data Type     |
|-------------|-------|-------------|----------------|---------------|
| <b>id</b>   | 1     | item id     | path           | string        |
| <b>data</b> |       | 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!

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

| Parameter   | Value                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>id</b>   | 1                                                                                                                                                                         |
| <b>data</b> | <pre>{   "date": "2019-02-11T18:12:34.433Z",   "reviewer_name": "Lucinda",   "reviewer_email": "lucy@think.org",   "comment": "bah",   "rating": 1,   "itemId": 1 }</pre> |

- \_\_ f. Click **Try it out**.

- \_\_ g. Confirm that the operation created the item review.

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

- \_\_\_ h. 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

## 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 last two exercises (Exercise 5 and 6). You must complete this lab exercise in the Ubuntu host environment.

## 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.

- 1. 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.

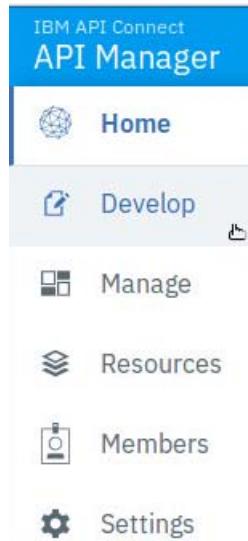


### Information

You can ignore any deprecation warnings.

- 2. View the generated `yaml` file.
  - a. Open `inventory.yaml` in **gedit**.
  - b. Review the `inventory.yaml` file.
- 3. 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:
    - Username: ThinkOwner
    - Password: Passw0rd!
 Click **Sign in**.  
You are signed in to API Manager.

- \_\_\_ 4. Click the tile Develop APIs and Products, or select the Develop option from the side menu.



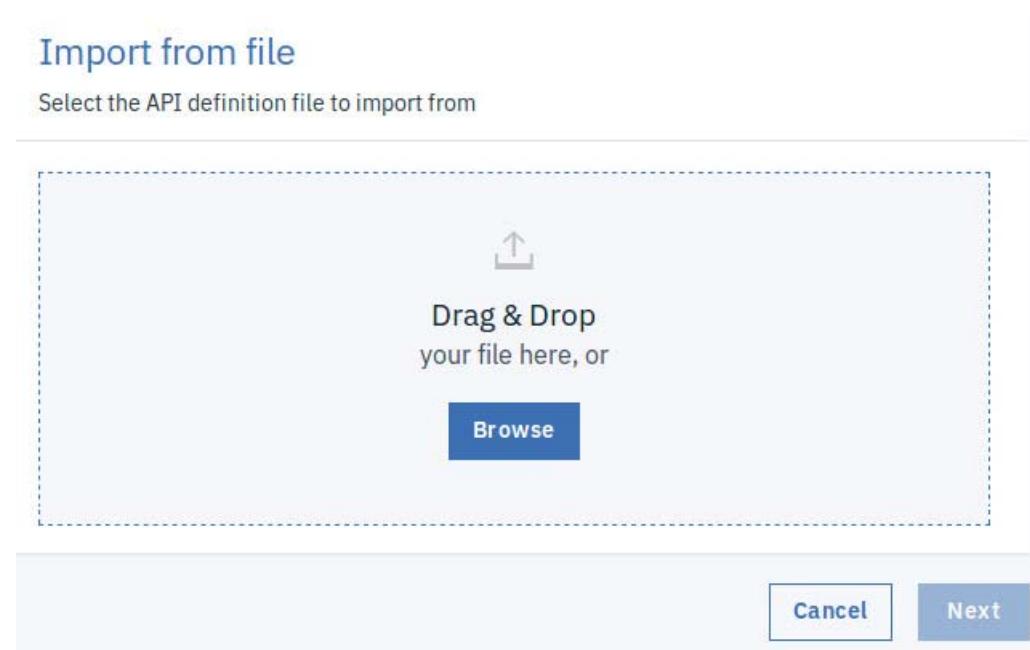
- \_\_\_ 5. Create an API definition by importing the `Inventory.yaml` file into API Manager
- \_\_\_ a. In the API Manager develop page, click **ADD**.  
Then, select **API**.
- \_\_\_ b. On the Add API page, select **Existing OpenAPI**.

The screenshot shows the 'Import' dialog box. It contains two options: 'From existing WSDL service (REST proxy)' and 'New OpenAPI'. Below these, there is a section titled 'Import' which contains a list of import sources. One source, 'Existing OpenAPI', is selected and highlighted with a red border. At the bottom of the dialog are 'Cancel' and 'Next' buttons.

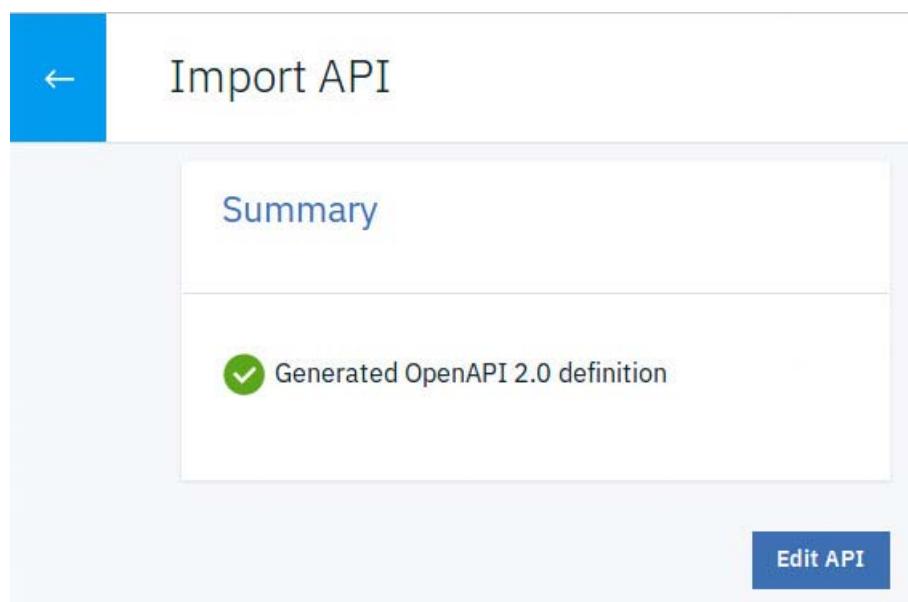
The selector is highlighted.

- \_\_\_ c. Click **Next**.

- \_\_\_ d. In the Import from file page, click **Browse**.



- \_\_\_ e. Go to the `/home/inventory` directory  
\_\_\_ f. Select the `inventory.yaml` file and click **Open**.  
\_\_\_ g. The YAML is imported and successfully validated.  
\_\_\_ h. Click **Next**.  
\_\_\_ i. Leave the Activate API option cleared. Click **Next**.  
\_\_\_ j. The generated OpenAPI 2.0 definition is created.



- \_\_\_ k. Click **Edit API**.  
The inventory API is displayed in the design view.  
\_\_\_ l. Scroll down with API Setup selected. Notice that the base path is set to `/inventory`.

- \_\_ m. Click the **Paths** category.

The screenshot shows the LoopBack API Designer interface. At the top, there are tabs for "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. The main content area is titled "Paths" and contains a table with the following data:

| NAME                      |
|---------------------------|
| /items/{id}/reviews/{fk}  |
| /items/{id}/reviews       |
| /items/{id}/reviews/count |
| /items                    |
| /items/replaceOrCreate    |
| /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.

- \_\_ n. Click the **Properties** category in the design view. Then, click **Add**.

The screenshot shows the LoopBack API Designer interface. The sidebar categories are the same as the previous screenshot. The main area is titled "Properties". It contains a table with the following data:

| PROPERTY NAME | ENCODED | DESCRIPTION                   | ⋮ |
|---------------|---------|-------------------------------|---|
| target-url    | false   | The URL of the target service | ⋮ |

An orange box highlights the "Add" button in the top right corner of the table header.

- \_\_\_ o. 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

### Default value (optional)

### Description (optional)

- \_\_\_ p. Click **Save**.
- \_\_\_ q. The app-server property is added to the list of properties for the inventory API.

## Properties

| PROPERTY NAME              | ENCODED | DESCRIPTION                               |
|----------------------------|---------|-------------------------------------------|
| <a href="#">target-url</a> | false   | The URL of the target service             |
| <a href="#">app-server</a> | false   | Host where the inventory application runs |

- \_\_\_ r. Click **target-url** in the properties list for the inventory application.

\_\_\_ s. 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

\_\_\_ t. 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 hostname 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.

- \_\_\_ 6. 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
```

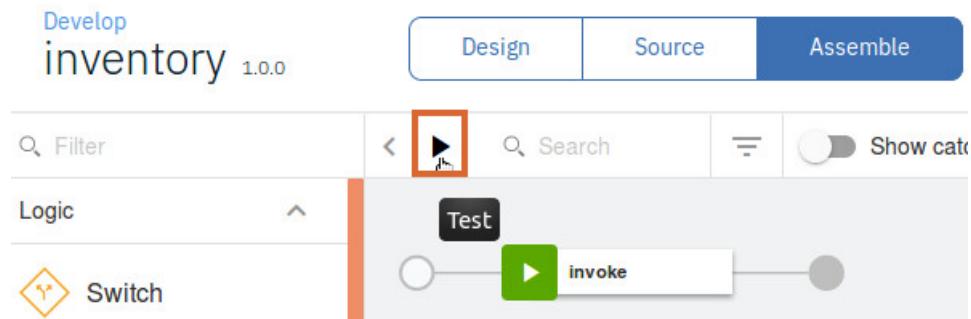
- \_\_\_ b. Start the Loopback application.

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

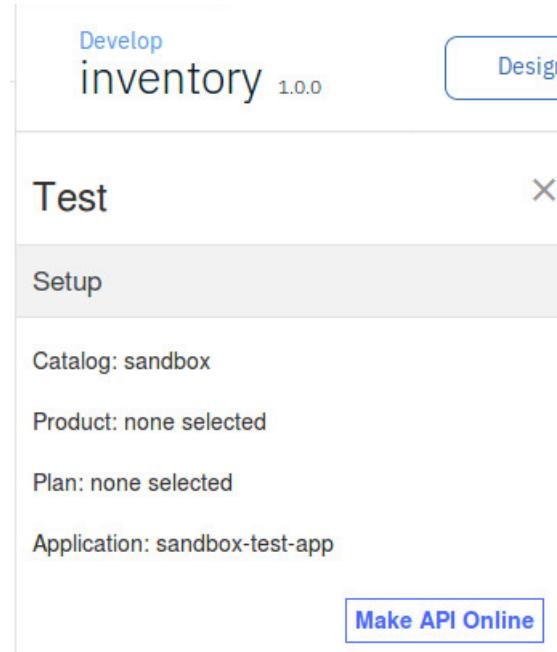
```
(node:7758) DeprecationWarning: current URL string parser is deprecated,
and will be removed in a future version. To use the new parser, pass
option { useNewUrlParser: true } to MongoClient.connect.
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- \_\_\_ 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.

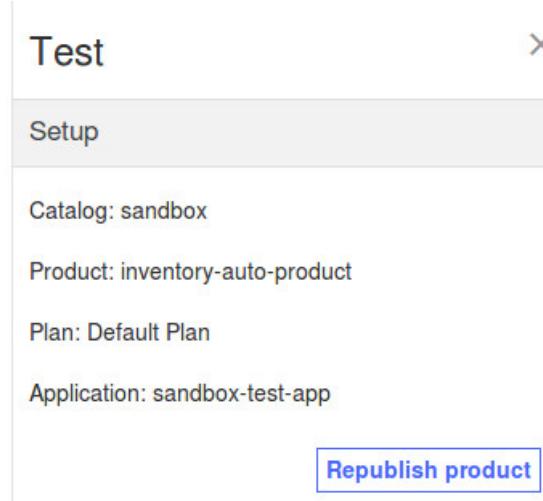


- \_\_\_ b. The test dialog opens.



Click **Make API Online**.

- \_\_\_ c. A Product and Plan are now added to the inventory API.



- \_\_\_ d. Scroll down to choose an operation to invoke.  
Select get /items from the operation drop-down.

The 'Operation' dropdown menu is open, showing the selected option 'get /items' highlighted with a red border.

parameters

Filter defining fields, where, include, order, offset, and limit - must be a JSON-encoded string  
({"something":"value"})

filter

- \_\_\_ e. Scroll down, then click **Invoke**.

The 'Invoke' configuration panel includes the following settings:

- Repeat
- Repeat the API invocation a set number of times, or until the stop button is clicked
- Stop after: 10
- Stop on error:
- Invoke** (button)

- \_\_\_ f. The test returns a response with status code 200 OK.

The screenshot shows the 'Test' interface of an API gateway. The 'Response' tab is selected. The status code is 200 OK. The response time is 642ms. The Headers section lists several HTTP headers, including apim-debug-trans-id, apiconnect, content-type, and x-ratelimit. The Body section shows a JSON array with one item, which has a name field set to "Dayton Meat Chopper".

```

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.



## Troubleshooting

If you get a 500 error, make sure the inventory application is running.

```
{
 "httpCode": "500",
 "httpMessage": "URL Open error",
 "moreInformation": "Could not connect
to endpoint"
}
```

- \_\_\_ 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                                                                                                                                                                                                                                                                                                                                               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Status code:<br/>200 OK</p> <p>Response time:<br/>271ms</p> <p>Headers:</p> <pre>apim-debug-trans-id: 426530149-Landlord- apiconnect-5accf6e1-5afc-494d-948f- 65556c19abe3 content-type: application/json; charset=utf-8 x-ratelimit-limit: name=default,100; x-ratelimit-remaining: name=default,98;</pre> <p>Body:</p> <pre>{ "count": 12 }</pre> |

\_\_ c. Close the test window.

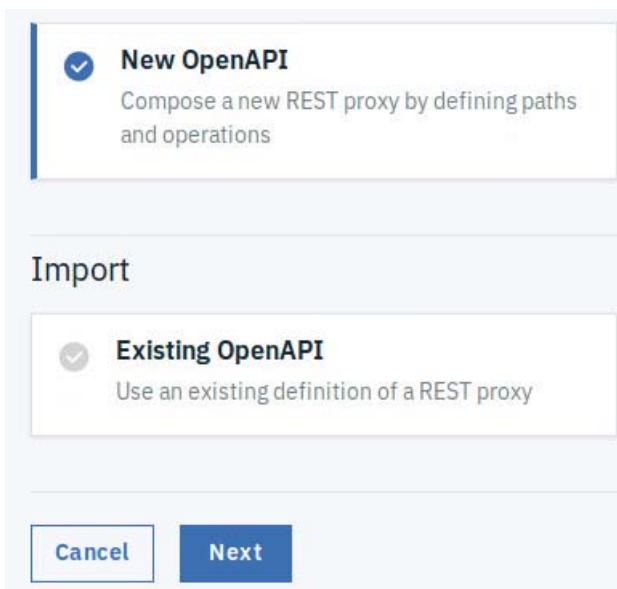
\_\_ d. Type **CTRL-C** to stop the inventory application in the terminal window.

You have successfully called the inventory application from the API gateway.

## 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. In this exercise, you 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**.

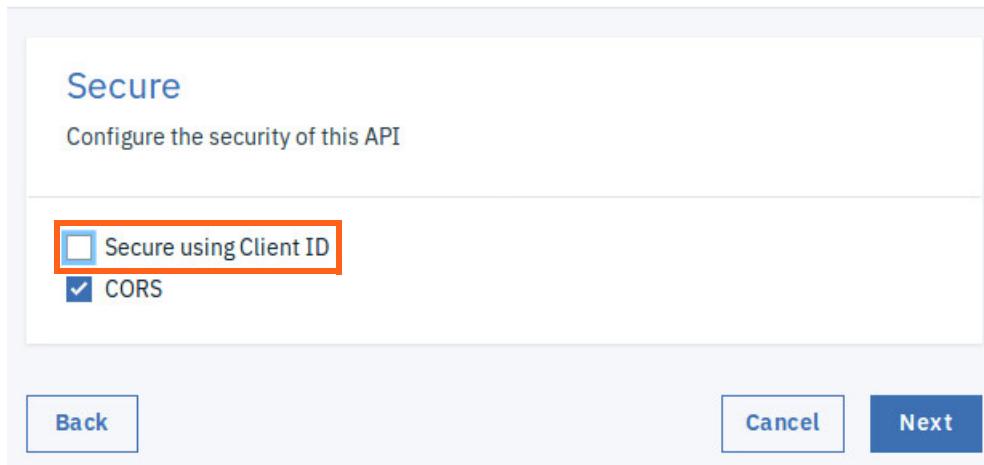


Click **Next**.

- \_\_\_ d. Type the following details for the `financing` API definition:
    - Title: `financing`
    - Name: `financing`
    - Version: `1.0.0`
    - Base path: `/financing`
- Click **Next**.

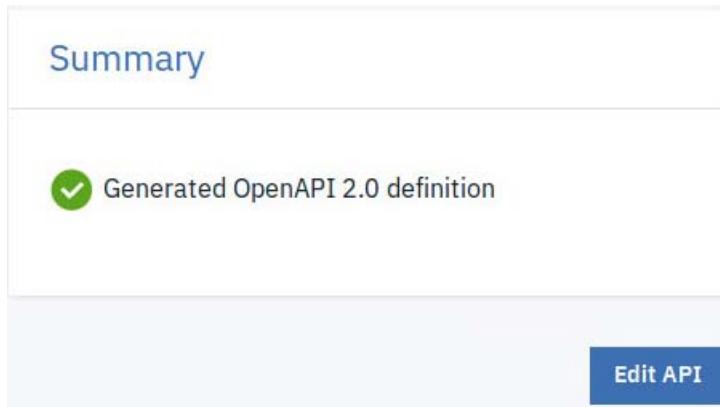
- \_\_\_ e. Clear the option “Secure using client ID” on the secure page of the create new OpenAPI definition.

## 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 radio buttons are shown: 'DataPower Gateway (v5 compatible)' and 'DataPower API Gateway', with the latter being selected. A yellow warning box 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.' At the bottom right are 'Cancel' and 'Save' buttons.

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 in the IBM API Connect developer console. The left sidebar has tabs: API Setup, Security Definitions, Security, Paths, Definitions (which is selected), and Properties. The main area has a title 'Definitions' and a sub-instruction: 'API request and response payload structures are composed using OpenAPI schema definitions.' Below is a table:

| NAME | TYPE | DESCRIPTION    |
|------|------|----------------|
|      |      | No items found |

- \_\_\_ b. Click **Add** to add a definition.
- \_\_\_ c. Set the name of the definition to: `paymentAmount`

The screenshot shows the 'Add Definition' dialog. It has three fields: 'Name' (value: paymentAmount), 'Type' (value: object), and 'Description (optional)' (empty). At the bottom right is a blue 'Add' button.

- \_\_\_ 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 passes as an input parameter.

In this section, you 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 with two main sections:

- Path name:** A text input field containing `/calculate`.
- Path Parameters:** A table with columns: REQUIRED, NAME, LOCATED\_IN, TYPE, DESCRIPTION, and DELETE. An "Add" button is located at the top right of this section.
- Operations:** A table with columns: NAME, and an "Add" button at the top right. Below the table, it says "No items found".



### 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 the operations area of the `/calculate` path, select the **Add** 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 to 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**

| REQUIRED                 | NAME | LOCATED IN    | TYPE          | DESCRIPTION | DELETE |
|--------------------------|------|---------------|---------------|-------------|--------|
| <input type="checkbox"/> |      | Choose one... | Choose one... |             |        |
| <input type="checkbox"/> |      | Choose one... | Choose one... |             |        |
| <input type="checkbox"/> |      | Choose one... | Choose one... |             |        |

- \_\_\_ h. Set the parameters according to the table of values.

Table 4. `GET /calculate` operation parameters

| Required | Name     | Located in | Type  | Description              |
|----------|----------|------------|-------|--------------------------|
| Yes      | amount   | query      | float | amount to finance        |
| Yes      | duration | query      | int32 | length of term in months |
| Yes      | rate     | query      | float | interest rate            |

| 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 **Response** 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 schema to the paymentAmount model definition that you created earlier.

**Response**

| 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 with the following details:

- Header:** Develop financing 1.0.0
- Top Navigation:** Design (selected), Source, Assemble
- Left Sidebar (under API Setup):**
  - API Setup
  - Security Definitions
  - Security
  - Paths
  - Definitions
  - Properties
  - Target Services
  - Categories
  - Activity Log** (selected)
- Main Content Area:**

### Activity Log

Configure properties of the activity log

**Activity Log:** Off —  On

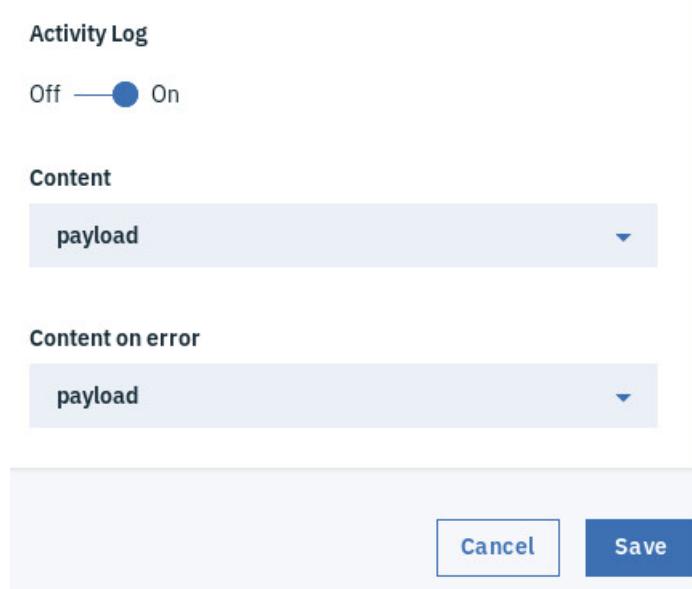
**Content:** activity

**Content on error:** header
- Bottom Buttons:** Cancel, Save

- 2. Set the activity log for the financing API.
  - a. Change the content type to payload.
  - b. A dialog is displayed to enable buffering. Click **Continue**.

\_\_\_ c. Select the following property values:

- Activity Log: **On**
- Content: **payload**
- Content on error: **payload**



\_\_\_ d. 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

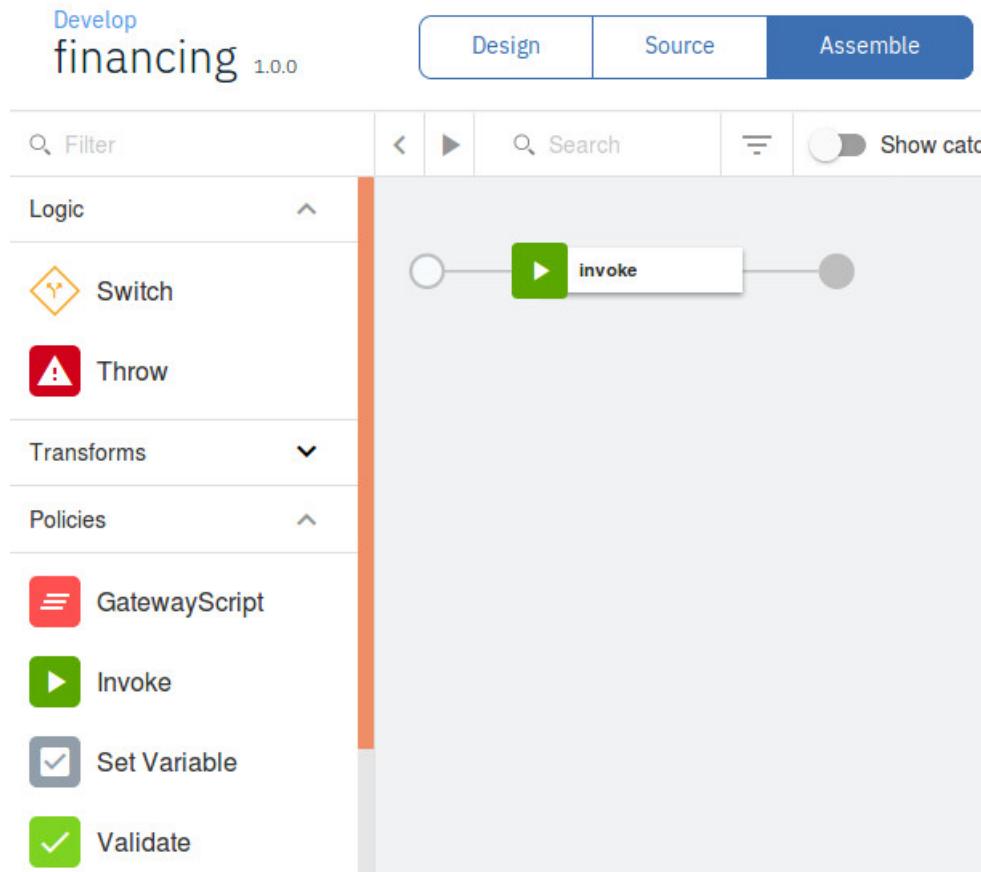
In the last section, you created an API operation that is named `GET /calculate` in the financing API. The operation takes three query parameters: the amount to finance, the length of the financing term in months, and the interest rate. The operation returns a JSON object of type `paymentAmount`: the monthly payment cost.

In this section, you assemble a message processing policy that calculates the payment amount. Then, you invoke a remote SOAP web service to calculate the payment amount and 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.



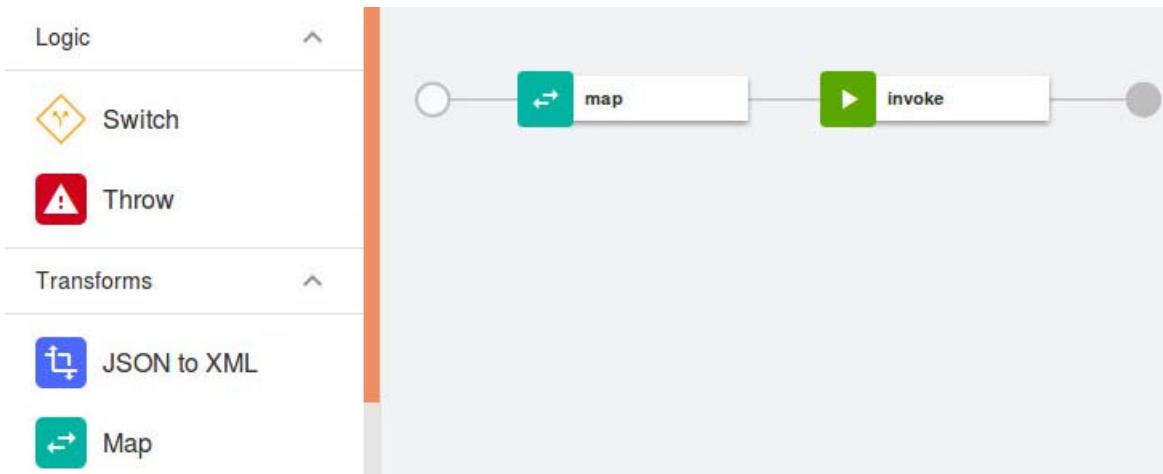
- 2. Since the DataPower Gateway was set as the default gateway for all catalogs, the list of API Gateway-supported policies are displayed in the palette.



An invoke policy is already added to the policy sequence in the free-form area.

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.

- \_\_\_ 3. 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.



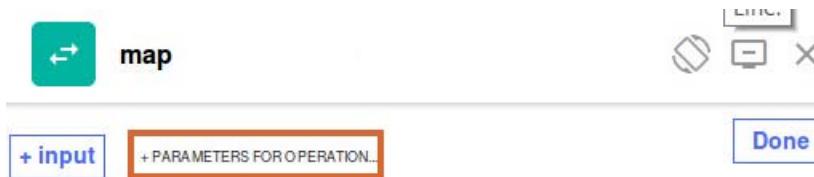
- \_\_\_ c. The properties for the map policy opens automatically when you drop the policy on the canvas.
- \_\_\_ d. Click the plus sign (+) icon to expand the map properties window.



- \_\_\_ 4. 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`.

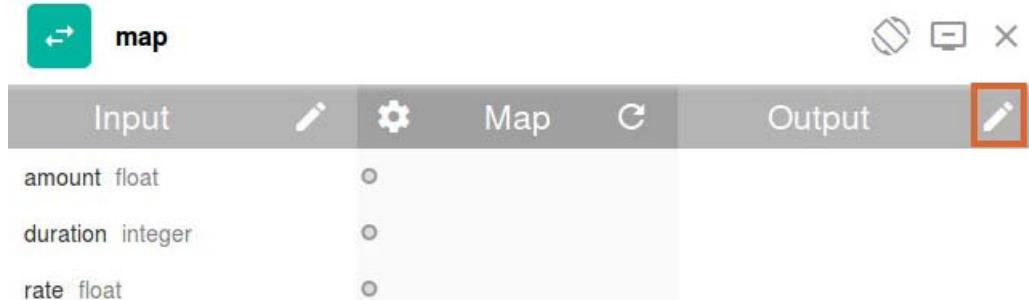
|                                          |              |
|------------------------------------------|--------------|
| Context variable                         | Name         |
| <code>request.parameters.amount</code>   | amount       |
| Content type                             | Definition * |
|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

- \_\_\_ e. Confirm that the input parameters match the following table.

Table 5. Financing API calculate operation input parameters

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

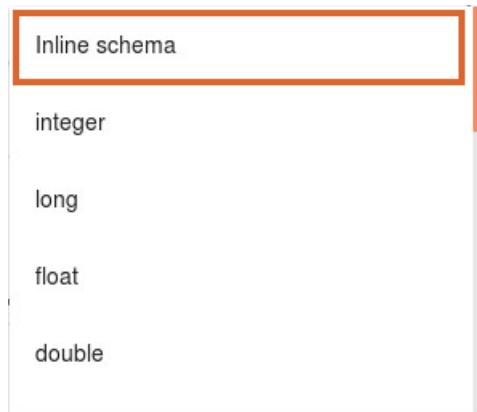
- \_\_\_ f. Click **Done**.
- \_\_\_ 5. Edit the **Output** parameters of the map policy.
- \_\_\_ a. In the map policy editor, click the edit outputs icon from the **Output** column.



- \_\_\_ b. Click **+ output** to add an output parameter.
- \_\_\_ c. Leave the context variable set to `message.body`.
- \_\_\_ d. Set the Content type to `application/xml`.



- \_\_\_ e. In the **Definition** field, scroll up and select `inline schema`.



**Note**

Leave the inline schema editor open in your web browser. To avoid typing errors, you copy the output parameters of the map policy action from a sample schema file later in the next step.

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. 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. Right-click and select **Select All**.
  - \_\_\_ e. Right-click and select **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

```

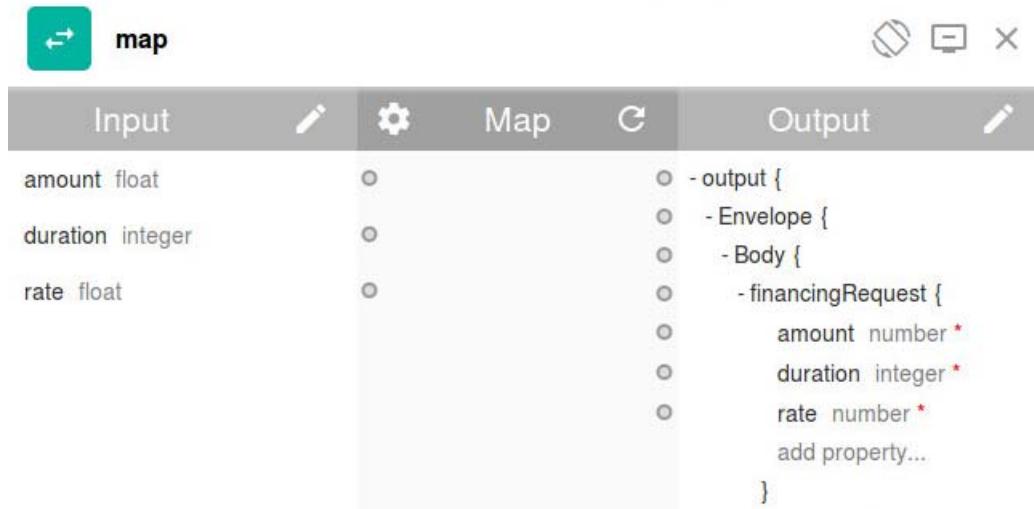
< Schema as YAML Schema as JSON Generate from sample JSON >
26
27 title: 'output'
28 type: object
29 id: 'http://services.think.ibm/envelope/body/financingRequest/rate'
30 properties:
31 rate:
32 type: number
33 name: rate
34 required:
35 - amount
36 - duration
37 - rate
38 additionalProperties: false
39 financingRequest:
40 type: object
41 name: financingRequest
42 required:
43 - financingRequest
44 additionalProperties: false
45 Body:
46 type: object
47 name: Body
48 required:
49 - Envelope
50 additionalProperties: false
51 Envelope:
52 type: object
53 name: Envelope
54 required:
55 - output
56 additionalProperties: false
57 title: output
58

```

done cancel

- \_\_\_ h. Click **done** in the inline schema editor.
- \_\_\_ i. Click **Done** in the **output** parameters editor.

\_\_\_ 7. Review the **map** policy action.



### Information

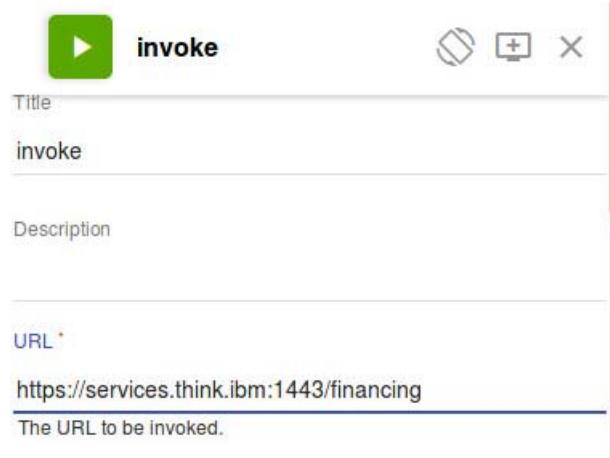
On the left side, the input to the GET /calculate API operation consists of three query variables: `amount`, `duration`, and `rate`. On the right side, the SOAP web service accepts three parameters in the XML SOAP request message.

In the next step, you 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.

- \_\_\_ 8. Map the `amount`, `duration`, and `rate` API input parameters to the SOAP web service request.
- Click the node (circle) that follows the `amount` input parameter.
  - Click the node (circle) at the `amount` parameter in the Output column.
  - Repeat the process to connect the `duration` and `rate` parameters to the output parameters of the same name.



- \_\_\_ 9. Close the map policy editor.
- \_\_\_ 10. 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. Change the HTTP method from Keep to **POST**.
- \_\_\_ d. Close the invoke policy editor.
- \_\_\_ e. Save the API.
- \_\_\_ 11. 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');
```

The screenshot shows the 'gatewayscript' editor window. At the top, there's a red icon with three horizontal lines and the word 'gatewayscript'. To the right are icons for saving, closing, and exiting. Below the title bar, there are two sections: 'Title' containing 'gatewayscript' and 'Description' which is empty. In the main area, there's a code editor with a dark background and white text. A single line of code is highlighted in yellow: '1 context.message.header.set('Content-Type', 'application/xml');'. The number '1' is in green, indicating it's the first line of code.

- \_\_\_ d. Close the **GatewayScript** editor.
- \_\_\_ 12. Add a **Parse** policy after the gateway script policy. Close the parse properties window without changes.
- \_\_\_ 13. Add an **XML to JSON** policy to convert the SOAP service response to a JSON message *after* the parse policy.



Close the XML to JSON properties window.

- \_\_\_ 14. 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.

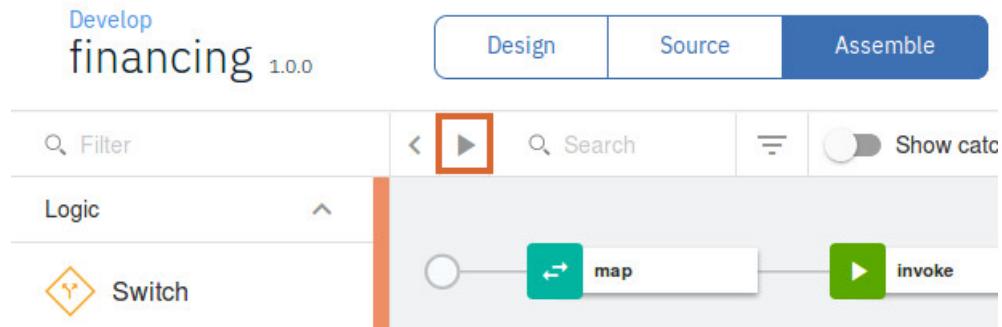
- The **map** policy copies the `amount`, `duration`, and `rate` query parameters into a SOAP request message. These three input parameters map to XML elements of the same name.
- The **invoke** policy makes a SOAP service request, and captures the response message.
- The **gatewayscript** policy takes 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.
-

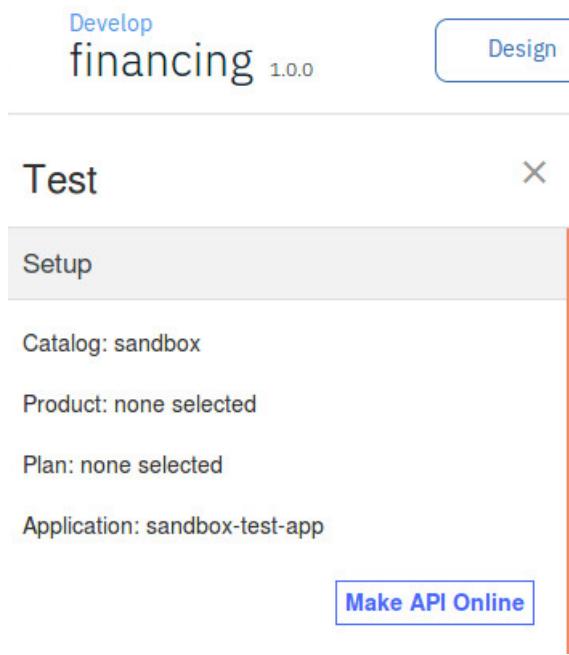
## 7.7. Test the financing APIs by calling it on the gateway

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.



The test dialog opens.



- c. 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                           |
| <br><a href="#">Generate</a>  |
| term in months<br>duration *  |
| 24                            |
| <br><a href="#">Generate</a>  |
| interest rate<br>rate *       |
| 0.05                          |

f. Click **Invoke**.

g. Confirm that the financing operation returns a 200 OK status code.

| Response               |
|------------------------|
| Status code:<br>200 OK |
| Response time:<br>74ms |

- h. 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"
 }
 }
 }
}
```

- i. You successfully called the financing application from the API gateway.  
 j. Close the test window.



### Information

If you do not get a successful response from the call, see the **Troubleshooting Issues** section in Appendix B.

## 7.8. Create the logistics API definition

In this section, you define a third API named `logistics`. This API provides two REST operations:

- 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.

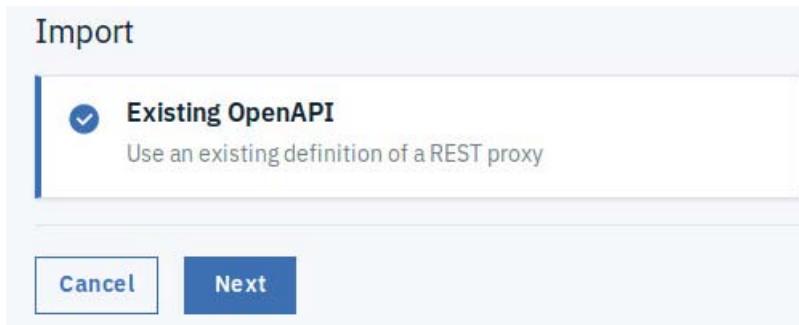
In the interest of time, you import a predefined OpenAPI definition file, `logistics_1.0.0.yaml`. You complete the API definition by creating an assembly flow for the `GET /stores` API operation. The `GET /shipping` operation is not completed in this exercise.

- 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

| 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     |       |

- 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. Your list might show APIs from prior exercises.

| 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 in 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**.  
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. Verify there are no security definitions listed.

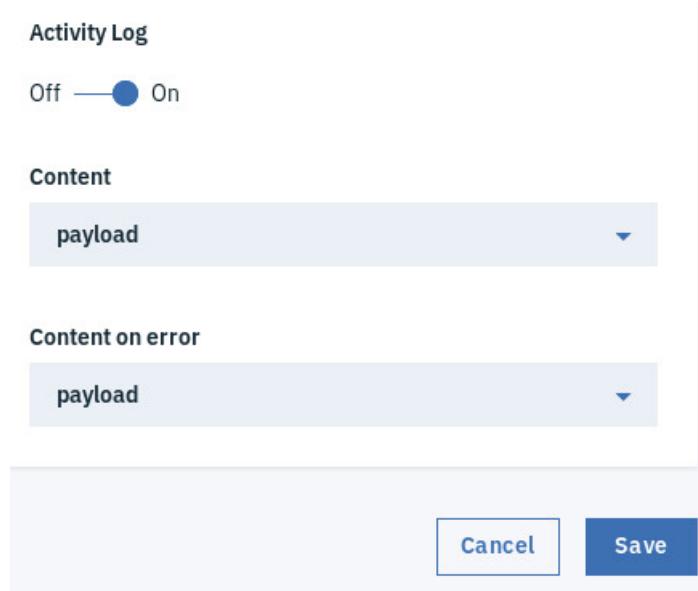
- \_\_\_ 4. In the Design view with the logistics API selected, click the **Activity Log** category.

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 subtitle 'Configure properties of the activity log'. It contains a section for 'Activity Log' with a switch that is set to 'On'. There are two dropdown menus under 'Content': one set to 'activity' and another set to 'header'. At the bottom right are 'Cancel' and 'Save' buttons.

- \_\_\_ 5. Set the activity log for the logistics API.
- \_\_\_ a. Change the Activity Log from **Off** to **On**.
- \_\_\_ b. When you change the content type to payload, a dialog is displayed to enable buffering. Click **Continue**.

\_\_\_ c. Select the following property values:

- Activity Log: **On**
- Content: **payload**
- Content on error: **payload**



\_\_\_ 6. Click **Save**.

## 7.10. Define the message policies for the GET /stores API operation

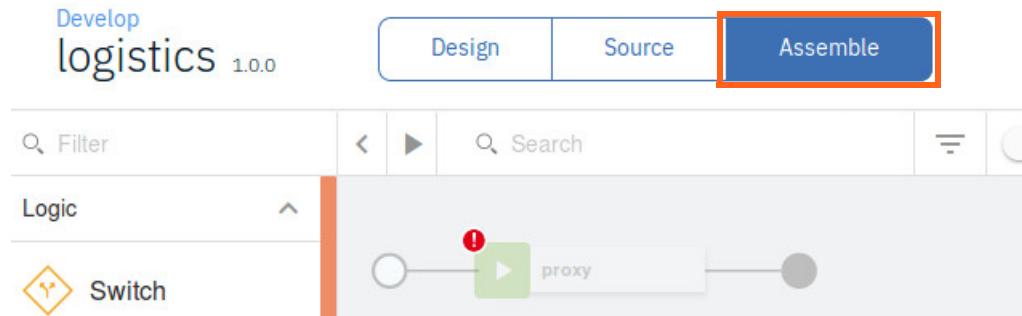
The logistics API defines two API operations: GET /shipping and GET /stores.

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. You also add a gateway script policy to format the contents of the API response message. The API calls a remote geolocation service to find a store location based on the client's postal code.

- 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.



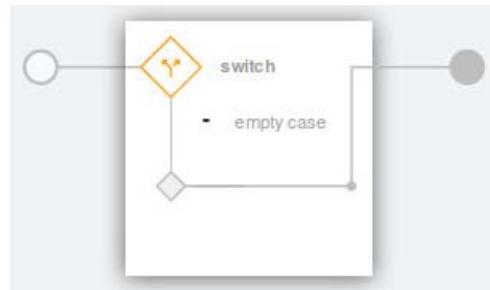
- 2. Hover over the proxy policy until the **delete** icon is displayed.



Click to delete the proxy policy.

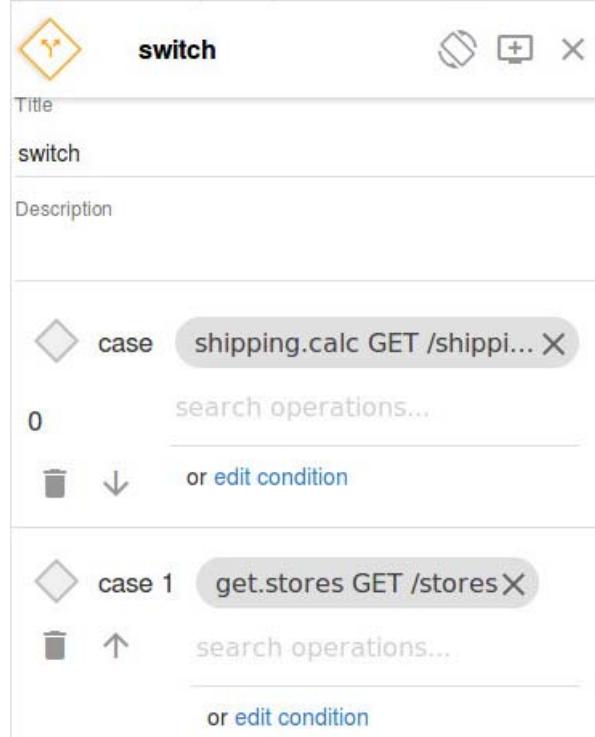
Only the start and end sequences remain.

- 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.

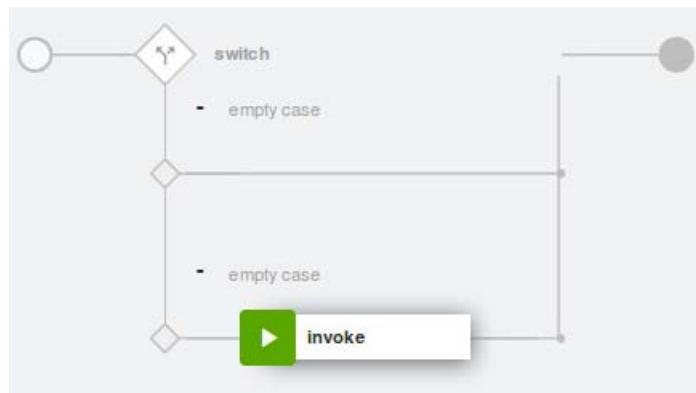


- c. Ensure that the **Switch** policy is open in the Properties view.

- d. Select the search operation field in the **case 0** branch.
- e. Select shipping.calc GET /shipping.
- f. Click **+ Case** to create a case for the operation switch.
- g. In **case 1**, select get.stores GET /stores as the operation.



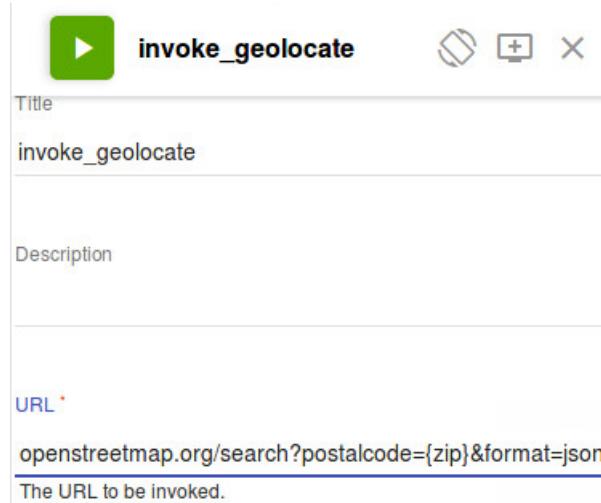
- 4. Close the properties view.
- 5. Add an **invoke** policy to the case 1 condition in the switch.
- a. In the GET /stores case in the switch policy, place an invoke policy.



- 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>



- Note: The URL that is coded above includes the zip parameter that is surrounded by curly brackets.
- Scroll to the bottom and clear Stop on error.
- Response object variable: **geocode\_response**



## 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 (get.store GET /stores).

\_\_ 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];

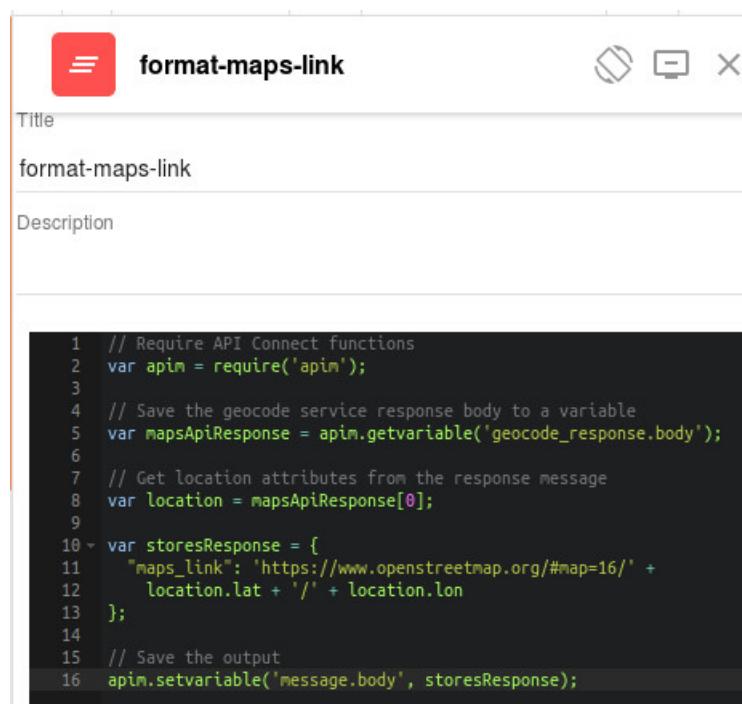
var storesResponse = {
 "maps_link": 'https://www.openstreetmap.org/#map=16/' +
 location.lat + '/' + location.lon
};

// 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.



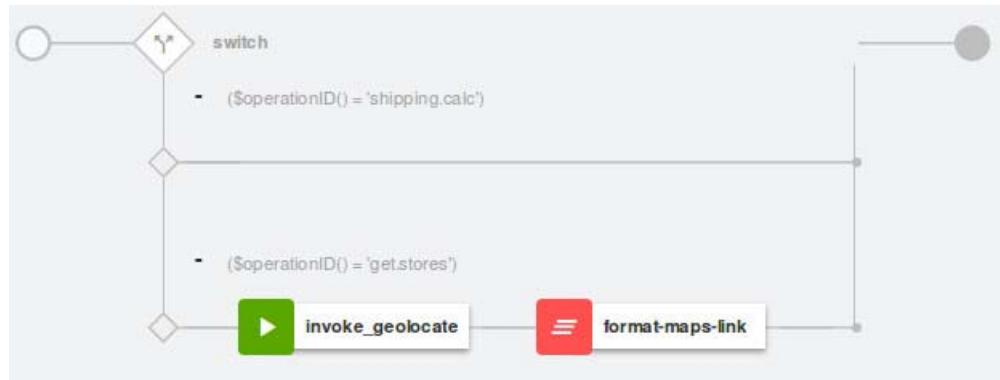
```
1 // Require API Connect functions
2 var apim = require('apim');
3
4 // Save the geocode service response body to a variable
5 var mapsApiResponse = apim.getvariable('geocode_response.body');
6
7 // Get location attributes from the response message
8 var location = mapsApiResponse[0];
9
10 var storesResponse = {
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

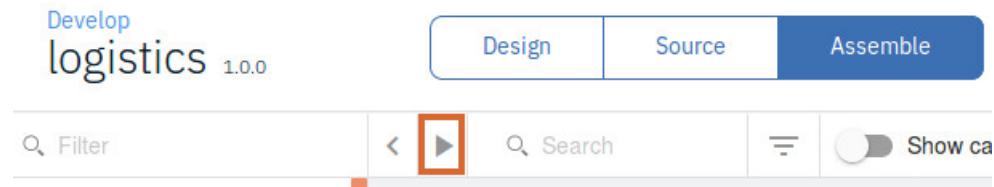
In the case one branch of the operation switch construct, the `GET /stores` API operation retrieves the map coordinates based on the US postal code (Zip code) from the API request. The gateway script policy takes the latitude (`location.lat`) and longitude (`location.lon`) values to build an Open Street Map link. The `GET /shipping` branch is a null operation.



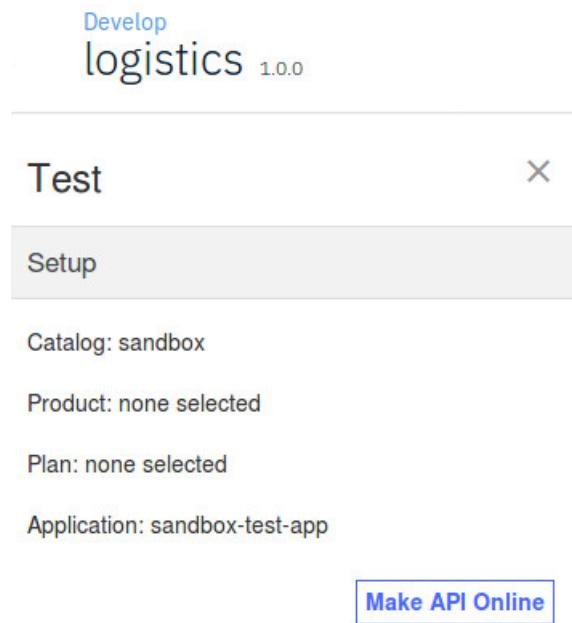
## 7.11. Test the stores API by calling them on the gateway

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. Verify you are on the **Assemble** page of the `logistics-1.0.0` API.
  - 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 /stores` operation.
- e. Type the following values for the `get /calculate` REST API parameters:

- zip: 15222

**Operation**

Choose an operation to invoke:

Operation  
get /stores

**parameters**

zip \*  
15222

- f. Click **Invoke**.
- g. Confirm that the logistics stores operation returns the coordinates of the map as a JSON object in the HTTP response message.

**Response**

Status code:  
200 OK

Response time:  
601ms

Headers:

```
apim-debug-trans-id: 426530149-Landlord-
apiconnect-d4036a41-f06b-4a55-880a-
65556c196ab7
content-type: text/xml
x-global-transaction-
id: 196c55655c6c8f3500009452
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,97;
```

Body:

```
{
 "maps_link": "https://www.openstreet
map.org/#map=16/40.44048745/-79.999033
7281745"
}
```

Verify the response body returns the map coordinates that correspond to the postcode that you chose. The coordinates might not be exact.



## Troubleshooting

If you get an error when testing an API, it might be due to the two virtual machines being out of sync. Refer to the **Troubleshooting Issues** section in Appendix B.

---

## End of exercise

## Exercise review and wrap-up

In the first part of this exercise, you examined how to build a REST-to-SOAP bridge at the API gateway. Then, 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.

# 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 last three exercises (Exercises 5, 6, and 7). You must complete this lab exercise in the Ubuntu host environment.

## 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, you 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 then 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:
    - Username: 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**.

### Resources

| TITLE | TYPE           |
|-------|----------------|
|       | 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**.

**Supported grant types**

Implicit  
 Application  
 Access code  
 Resource owner password

**Supported client types**

Confidential  
 Public

**Buttons:** Back, Cancel, Next

- \_\_\_ d. Leave the default values for the supported client types (Confidential). Click **Next**.
- \_\_\_ e. On the Scopes page, replace the sample values and type:
- Name: **inventory**
  - Description: **Access to the inventory API**

**Scopes**

Add scopes for this OAuth Provider.

| NAME      | DESCRIPTION                 | DELETE |
|-----------|-----------------------------|--------|
| inventory | Access to the inventory API |        |

**Buttons:** 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**
- \_\_ h. In the authentication area, click **Create Sample User Registry**.

The screenshot shows the 'Identity Extraction' configuration page. It has three main sections: 'Identity Extraction', 'Authentication', and 'Authorization'.  
In the 'Identity Extraction' section, under 'Collect credentials using', the value 'Basic Authentication' is selected.  
In the 'Authentication' section, under 'Authenticate application users using', the value 'No LDAP or authentication URL user registries found.' is selected. The button 'Create Sample User Registry' is highlighted with a red box.  
In the 'Authorization' section, under 'Authorize application users using', the value 'Authenticated' is selected.

- \_\_\_ i. A SampleAuthURL registry is created and added in the authentication drop-down list.

The screenshot shows three stacked configuration sections:

- Identity Extraction:** Shows "Collect credentials using" with a dropdown menu containing "Basic Authentication".
- Authentication:** Shows "Authenticate application users using" with a dropdown menu containing "SampleAuthURL", which is highlighted with an orange border.
- Authorization:** Shows "Authorize application users using" with a dropdown menu containing "Authenticated".

- \_\_\_ j. Click **Next**.

- \_\_ k. 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.

- \_\_\_ 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 'Edit Native OAuth Provider' interface. The left sidebar has tabs: Info (selected), Configuration, Scopes, User Security, Tokens, Token Management, Introspection, Metadata, OpenID Connect, and API Editor. The main area has fields: Name (oauth-provider), Description (optional) (empty), Gateway version (6000), Base path (/oauth20), and a checked checkbox labeled 'Enable debug response headers'. Below the form are 'Cancel' and 'Save' buttons.

Click **Save**.

- \_\_\_ b. Click the **Configuration** tab.

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             |                                               |                     |        |
| OAuth Providers | <b>SampleAuthURL</b>                          | Authentication URL  | :      |
|                 | <a href="#">Sandbox Catalog User Registry</a> | Local User Registry | :      |

- \_\_\_ c. In the Authentication URL User Registry dialog, type:

- URL: `https://services.think.ibm:1443/authorize`

- \_\_\_ d. Click **Save**.

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**.

### Authentication

Authenticate application users using (optional)

**SampleAuthURL**

Authentication URL (optional)

### Authorization

Authorize application users using

**Authenticated**

Cancel
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** in the navigation bar on the left.
  - \_\_\_ c. Click **OAuth Providers**. Then, click **Edit**.

Manage / Sandbox  
Settings

**OAuth Providers**

Manage the OAuth Providers configured for API Manager

| TITLE | TYPE           |
|-------|----------------|
|       | No items found |

Overview  
Gateway Services  
Lifecycle Approvals  
Roles  
Onboarding  
API User Registries  
**OAuth Providers**

- \_\_\_ d. Select oauth-provider.

|                                     | TITLE          | TYPE   |
|-------------------------------------|----------------|--------|
| <input checked="" type="checkbox"/> | oauth-provider | Native |

**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                                                                                                                                    |
|---------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 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. |

- \_\_\_ 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 username 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.

| APIs and Products |                 |
|-------------------|-----------------|
| TITLE             |                 |
|                   | financing-1.0.0 |
|                   | inventory-1.0.0 |
|                   | logistics-1.0.0 |

- 2. Create an OAuth security definition.
  - a. Click **Security Definitions**. Then, click **Add**.

| Security Definitions                                                                                                                                              |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Security definitions control client access to API endpoints, including API key validation, application user authentication, and OAuth. <a href="#">Learn more</a> |  |
| <b>Add</b>                                                                                                                                                        |  |

- 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>                   | <code>oauth</code>                                                                                |
| <b>Description (optional)</b> | <code>OAuth authorization settings for the inventory API</code>                                   |
| <b>Type</b>                   | <input type="radio"/> API Key <input type="radio"/> Basic <input checked="" type="radio"/> OAuth2 |
| <b>OAuth Provider</b>         | <code>oauth-provider</code>                                                                       |
| <b>Flow</b>                   | <code>Resource owner password</code>                                                              |
| <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 **Security** 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". At the bottom right is a blue "Save" button.

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>.

## 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 displayed.

| TITLE                        | CLIENT ID                        |
|------------------------------|----------------------------------|
| Sandbox Test App 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.

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.



### Hint

When copying between the web page and gedit, be sure to not click outside the Credentials dialogue otherwise new credentials will be generated.

You copy these values from the `inventory-cred` file later.

- \_\_\_ j. Click **Create**.

- \_\_\_ k. The credential is added.

| Credentials                          |                                  | Add |
|--------------------------------------|----------------------------------|-----|
| TITLE                                | CLIENT ID                        |     |
| 6ffa338f-af33-472a-89da-47fef3bc3abf | f061d5701fef6a75fe210a3dfc9b7d0f | ⋮   |
| Sandbox Test App Credentials         | 27aa227243f4aaead72e18757f77de48 | ⋮   |

## 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.

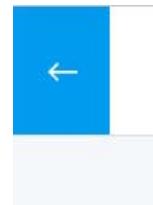
```
$ npm start
> inventory@1.0.0 start /home/localuser/inventory
> node .

(node:1188) 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
```

\_\_ 2. Open the inventory API in the editor.

\_\_ a. Click the **return arrow** in the upper left corner twice.



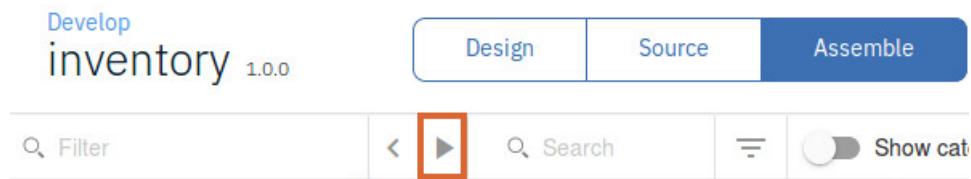
\_\_ b. Click the **Develop** option in the navigation menu.

\_\_ c. Click the **Inventory** API in the list of APIs and Products.

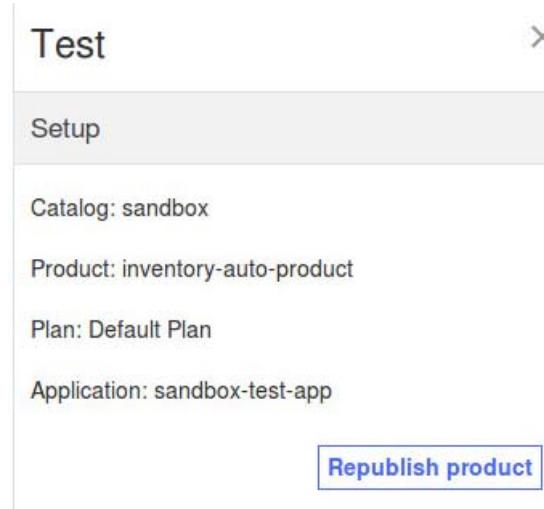
\_\_ 3. Test the inventory API in the assembly test feature.

\_\_ 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

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**

You must provide the explorer access token.

- \_\_\_ h. To build the curl command to retrieve the access token, open the template `~/lab_files/authorization/oauth.txt` to edit the curl command.
- \_\_\_ i. Replace the `<app_client_id>` and `<app_client_secret>` variables in the curl command template with the client ID and client secret you saved earlier.
- \_\_\_ j. Open a new terminal window.
- \_\_\_ k. After replacing the variables, paste the updated command into the terminal.

```
curl -k https://apigw.think.ibm/think/sandbox/oauth2/token -d
"grant_type=password&scope=inventory&username=user&password=pass&client_id=<app_client_id>&client_secret=<app_client_secret>"
```

- \_\_\_ l. Copy the access token from the terminal window (everything between the quotation marks). Paste the value into a document first to verify you copied the value.

```
localuser@ubuntu:~$ curl -k https://apigw.think.ibm/think/sandbox/oauth2/token -d
"grant_type=password&scope=inventory&username=user&password=pass&client_id=f061d5701fef6a75fe210a3dfc9b7d0f&client_secret=5724e0592459123429ddeb574686d0be"
{"token_type":"Bearer","access_token":"AAIgZjA2MWQ1NzAxZmVmNmE3NWZLMjEwYTNkZmM5YjdkMGa8cXveF7JX0jAMPZMe744NMe4v4bU_vDXz5PVc3rwM1ZBnGcbPHKloufiTz8_la5BX7zqJs_TShCZTfr5WJ0Op","scope":"inventory","expires_in":3600,"consented_on":1551221058}localuser@ubuntu:~$
```

- \_\_\_ m. Then, paste the token into the test feature in the explorer access token area.

`api_security_oauth_token_url:`  
`https://$(api.endpoint.address)/oauth20/oauth2`  
`/token`

`explorer_access_token`  
  
.....

- \_\_\_ n. Scroll down in the test feature. Then, click **Invoke**.



## Troubleshooting

If you receive a 404 error, try republishing the API and retrying the curl command.

```
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=b0ae7c84d244cf744633638fa61222f6&client_secret=1b7b81d0941227bbdf034f290014361c"
{"httpCode":404,"httpMessage":"Not Found","moreInformation":"API not found for requested URI"}localuser@ubuntu:~$
```

If you continue to get an error, this may be due to the two virtual machines being out of sync. Refer to the **Troubleshooting Issues** section in Appendix B.

- o. The response message is displayed in the test feature.

The screenshot shows a test result from a REST API testing tool. At the top right is a blue "Invoke" button. Below it is a "Response" section with the following details:

- 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, manufacturing began in 1927."}]
```

- p. 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. Close all terminals.
  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.

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 this exercise explained how to create and publish an OAuth 2.0 Provider: a set of API operations that authorize access to protected resources. You then created 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.

---

# 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 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.



## Information

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 last four exercises (Exercises 5, 6, 7, and 8). You must complete this lab exercise in the Ubuntu host environment.

## 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 .


```

```
(node:1188) 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
```

\_\_\_ d. Wait until the web server is started and listening (Web server listening...).



### Information

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, right-click and select **New 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
Strict-Transport-Security: max-age=0; includeSubDomains
X-Download-Options: noopen
X-Content-Type-Options: nosniff
Content-Type: application/json; charset=utf-8
Content-Length: 10474
ETag: W/"28ea-zVF7pZHtRnKWXp9ejxduQnmyLcY"
Date: Sat, 25 Jan 2020 23:42:15 GMT
Connection: keep-alive
```



### Information

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.

- 
- \_\_\_ 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. Close the terminal that was running the application.

## 9.2. Set up the Docker image configuration

With Docker technology, you can publish Loopback applications to a containerized environment. 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, if one is not already open.  
\_\_\_ 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
```

**Attention**

**Do not perform the steps on this page. This is for your reading purposes only.**

The docker build and docker pull commands do not work and will corrupt the lab in this VM due to the system date being set to January 25, 2020. Note in a production environment, these steps can be followed to build a docker image. Review steps e and f in this section but do not perform them, then proceed to step g and complete the lab.

- \_\_\_ e. Build the Docker image by using the Dockerfile

**Do not perform this step. This is for your reading purposes only.**

```
$ 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.g
z
(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
```

- \_\_\_ f. Pull an existing image by entering the following command:

```
docker pull kevinom/apic-inventory-image:2018.4.1.1
```

**Do not perform this step. This is for your reading purposes only.**

- \_\_\_ g. Display the list of Docker images.

```
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
alpine 3.8 c8bcc0af957 2 days ago 4.41MB
kevinom/apic-inventory-image 2018.4.1.1 38ad0838a12e 11 months ago 113MB
```

- \_\_\_ 2. Review and then remove the Docker image

- \_\_\_ a. Open the image

```
$ docker run -it kevinom/apic-inventory-image:2018.4.1.1 /bin/sh
```

- \_\_\_ b. View the list files

```
/inventory # ls
Dockerfile common node_modules package.json
client inventory.yaml package-lock.json server
```

- \_\_\_ c. View the node version

```
/inventory # node -v
v8.14.0
```

- \_\_\_ d. Exit the image

```
/inventory # exit
```

- \_\_\_ e. Obtain the container ID of the apic-inventory-image.

```
$ docker ps -a
CONTAINER ID IMAGE COMMAND
CREATED STATUS PORTS
NAMES
9956dfe358f4 kevinom/apic-inventory-image:2018.4.1.1 "/bin/sh"
About a minute ago Exited (0) 6 seconds ago
stupefied_blackwell
5563f02465c9 c8bcc0af957 "/bin/sh -c
'apk add..'
15 minutes ago Exited (1) 15 minutes ago
jovial_proskuriakdockerova
```

- \_\_\_ f. Remove the docker image by replacing the <container ID> in the command below with the container ID listed in the command response above.

```
$ docker rm <container ID>
```



## Information

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 `apic-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.

- \_\_\_ 3. Run the application on the Docker image.

- \_\_\_ a. Return to the home directory.

```
$ cd ~
```

- \_\_\_ b. In the terminal, type:

```
$ docker run --add-host platform.think.ibm:10.0.0.10 -p 3000:3000
kevinom/apic-inventory-image:2018.4.1.1
```

```
> inventory@1.0.0 start /inventory
> node .
```

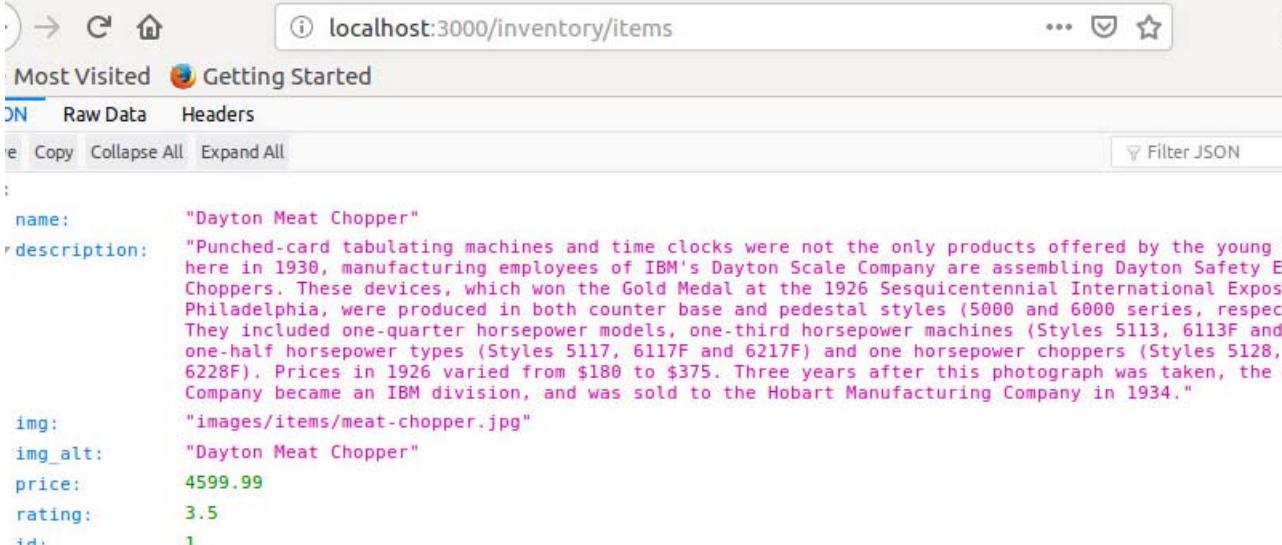
```
(node:21) DeprecationWarning: current URL string parser is deprecated,
and will be removed in a future version. To use the new parser, pass
option { useNewUrlParser: true } to MongoClient.connect.
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

- \_\_\_ 4. 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 showing a JSON response from `localhost:3000/inventory/items`. The JSON object represents an inventory item with the following fields and values:

```

{
 "name": "Dayton Meat Chopper",
 "description": "Punched-card tabulating machines and time clocks were not the only products offered by the young here in 1930, manufacturing employees of IBM's Dayton Scale Company are assembling Dayton Safety E Choppers. These devices, which won the Gold Medal at the 1926 Sesquicentennial International Expos Philadelphia, were produced in both counter base and pedestal styles (5000 and 6000 series, resp. They included one-quarter horsepower models, one-third horsepower machines (Styles 5113, 6113F and one-half horsepower types (Styles 5117, 6117F and 6217F) and one horsepower choppers (Styles 5128, 6228F). Prices in 1926 varied from $180 to $375. Three years after this photograph was taken, the 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
}

```

- \_\_\_ 5. 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          |
| 89a7bcac34c2       | kevinom/apic-inventory-image:2018.4.1.1 | "npm start"            |                |
| About a minute ago | Up About a minute                       | 0.0.0.0:3000->3000/tcp | brave_shockley |

- \_\_\_ c. In the terminal, type:

```
$ docker stop <container ID>
```



### Note

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.

- \_\_\_ 6. Close the terminal windows.  
\_\_\_ 7. Close the browser.

## End of exercise

## Exercise review and wrap-up

The first part of the exercise reviewed 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 then 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 last five exercises (Exercises 5,6,7,8 and 9). You must complete this lab exercise in the Ubuntu host environment.

# 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.

- \_\_\_ 1. 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.
- \_\_\_ 2. 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    |



## Note

Your list of APIs might show other APIs developed in the course.



## 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.

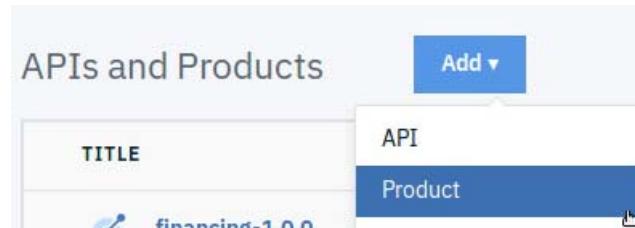
The logistics API was also created as a new OpenAPI definition in API Manager where the target URL for the stores points to an external service that returns the location of a US postal code.

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.

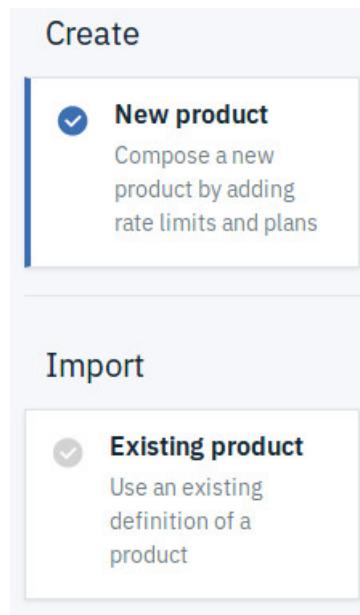
\_\_\_ 3. Create a single product for the APIs.

\_\_\_ a. On the Develop page, click **Add**.

Then, select **Product**.



\_\_\_ b. On the Add Product page, select **New Product**.



Click **Next**.

\_\_\_ 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



### Note

Your list of APIs might show other APIs developed in the course.

\_\_ e. Click **Next**.

\_\_ f. Accept the defaults to add a Default Plan to the product.

The screenshot shows a user interface for managing product plans. At the top, there's a header with the word 'Plans' and a blue 'Add' button. Below the header, a sub-header says 'Add plans to this product'. The main area is titled 'Default Plan'. It contains three fields: 'Title' with the value 'Default Plan', 'Description (optional)' with the value 'Default Plan', and 'Rate Limit' which is set to '100 / 1 hour'. At the bottom of the screen are three buttons: 'Back' (in a light blue box), 'Cancel' (in a light blue box), and 'Next' (in a dark blue box).

Click **Next**.

- \_\_ g. 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**

Click **Next**.

- \_\_ h. 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

- \_\_ i. Click the return arrow to return to the Develop page.  
 \_\_ j. The think-product is added to the list of APIs and products.

| APIs and Products            |            |               |
|------------------------------|------------|---------------|
| 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 |



### Note

Your list of APIs might show other APIs developed in the course.



## 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.

## 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.

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

- Header:** Develop, inventory 1.0.0, Design (selected), Source, Assemble.
- Left sidebar:** API Setup, Security Definitions, Security, Paths, Definitions, Properties (selected).
- Properties table:**

| PROPERTY NAME           | ENCODED | DESCRIPTION                               | ⋮ |
|-------------------------|---------|-------------------------------------------|---|
| <code>target-url</code> | false   | The URL of the target service             | ⋮ |
| <code>app-server</code> | false   | Host where the inventory application runs | ⋮ |

- c. Click `app-server` in the property list.

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
kevinom/apic-inventory-image:2018.4.1.1
```

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.

---

- d. Click **Cancel** to exit the editor without changes.
- e. 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 disabled the API key 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 API Management interface. At the top, there's a navigation bar with 'Develop' and 'financing 1.0.0'. Below it, three tabs are visible: 'Design' (which is selected), 'Source', and 'Assemble'. On the left, a sidebar has 'API Setup' and 'Paths' sections. The main area is titled 'Security' with the sub-section 'SECURITY DEFINITIONS'. A note says: 'Security definitions selected here apply across the API, but can be overridden for individual operations.' There's a blue 'Add' button. The 'Security' tab is highlighted with a red box.

- No security definitions are defined.
- \_\_\_ b. Click **Add**.
  - \_\_\_ c. Select the `clientIdHeader apiKey` security requirement.

The screenshot shows a modal dialog titled 'SECURITY DEFINITIONS'. It contains a single entry: 'clientIdHeader apiKey', with a checked checkbox next to it. A blue 'Save' button is at the bottom right.

- 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 navigation bar at the top has tabs for "Develop", "Design" (which is selected), "Source", and "Assemble". Below the navigation bar, there are sections for "API Setup" and "Security Definitions". A red box highlights the "Security" tab, which is currently selected. To the right of the "Security Definitions" section, there is a note: "Security definitions selected here apply across the API, but can be overridden for individual operations. [Learn more](#)". A blue "Add" button is located in the top right corner of this section. Below the "Security" tab, there are links for "Paths" and "SECURITY DEFINITIONS".

- No security definitions are defined.
- \_\_\_ b. Click **Add**.
  - \_\_\_ c. Enable the `clientIdHeader apiKey` security requirement.

The screenshot shows a modal dialog box titled "SECURITY DEFINITIONS". Inside the dialog, there is a list of security requirements: "clientIdHeader" and "apiKey", with a checkbox next to each. The checkbox for "clientIdHeader" is checked and highlighted with a red box. At the bottom right of the dialog is a blue "Save" button.

- \_\_\_ d. Click **Save**.  
The API is updated.
- \_\_\_ e. 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.

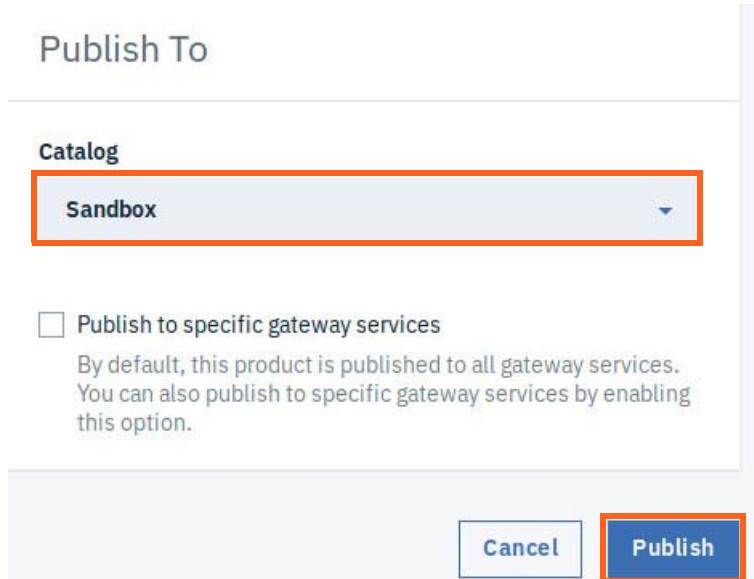
- 2. Publish the API product to the API Connect Cloud **Sandbox** catalog.

- a. In the list of APIs and Products, select the ellipsis option alongside the **think-product**. Then, select **Publish**.

| 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) | 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, make sure that the **Sandbox catalog** is selected and 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 the product 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 | ⋮ |
| ➤ think-product          | think-product 1.0.0          | Published | ⋮ |



### Note

Your list of products might show other products developed in the course.

- \_\_\_ 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 then 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.

# 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 by signing onto the Developer Portal as the owner of the consumer organization. You review the published products and APIs and then 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 in the API product by using the Developer Portal.

## 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

## 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.

## Requirements

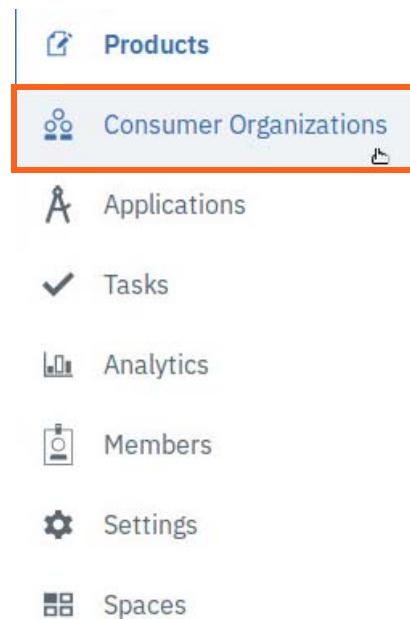
Before you start this exercise, you must complete the last six exercises (Exercises 5,6,7,8, 9, and 10). You must complete this lab exercise in the Ubuntu host environment.

## 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.

- \_\_\_ 1. 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:
    - Username: ThinkOwner
    - Password: Passw0rd!

Click **Sign in**.  
You are signed in to API Manager.
- \_\_\_ 2. 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.

[Manage](#) / Sandbox

## Consumer Organizations

| TITLE                     | OWNER                                    | STATE   |   |
|---------------------------|------------------------------------------|---------|---|
| Publish                   | OP<br>Owner Publish<br>owner@publish.org | Enabled | ⋮ |
| Sandbox Test Organization | TU<br>Test User                          | Enabled |   |

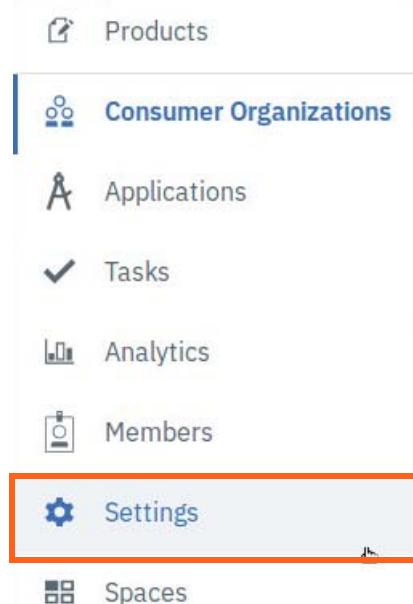


### Information

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.

- \_\_\_ 3. 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 main 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").

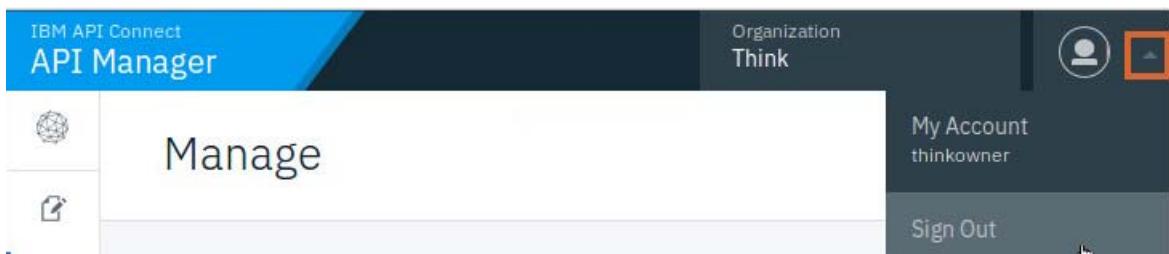
## Information

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.  
\_\_\_ 4. 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.

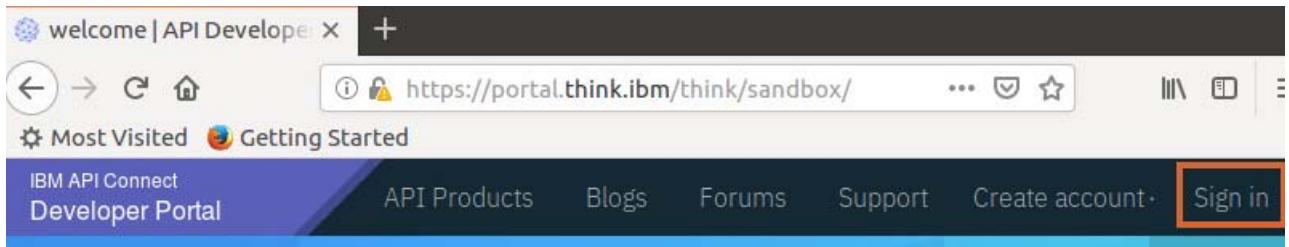
- \_\_\_ 1. Open the Developer Portal page.
  - \_\_\_ a. In a web browser address, type:  
`https://portal.think.ibm/think/sandbox`
  - \_\_\_ b. The Developer Portal public page is displayed.



### 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.

- Username: **OwnerPublish**
- Password: **Passw0rd!**

API Developer Portal

## Sign in

Sign in with Sandbox Catalog User Registry

Username

OwnerPublish

Password

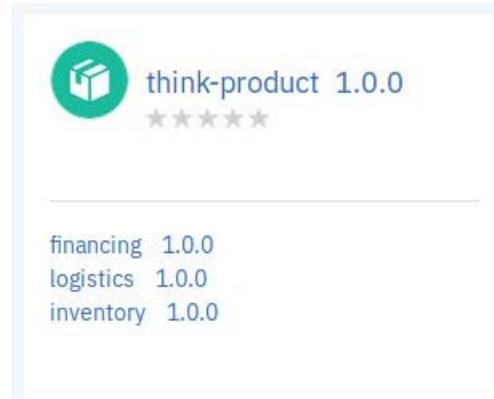
.....

Sign in

- \_\_\_ c. Click **Sign in**. 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 BM API Connect Developer Portal homepage. At the top, there's a purple header bar with the text "BM API Connect Developer Portal". Below it is a dark blue navigation bar with links for "API Products", "Apps", "Blogs", "Forums", and a search icon. On the right side of the navigation bar, there are buttons for "Organization" and "Publish". Below the navigation bar, the main content area has a large blue background featuring a white owl logo on the left and the text "brace yourselves. APIs are coming." in white. A purple button labeled "Explore API Documentation" is visible. At the bottom of the page, there's a white footer bar with the text "API Products" on the left and a red-bordered button labeled "See all products" on the right.

- \_\_\_ 4. The think-product that you published earlier is displayed in the list of published products.



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.



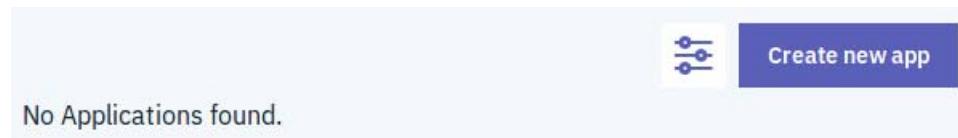
If you still do not see the product, you must publish the product. Refer to [Exercise 10, "Define and publish an API product,"](#) on page 10-1. Republish the product if necessary.

- \_\_\_ 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)

The screenshot shows a form for adding OAuth redirect URLs. It has a placeholder URL 'http://www.thinkapplication.com' and a red 'Add another item' button. At the bottom are 'Cancel' and 'Submit' buttons.

Click **Submit**.

The application is created.



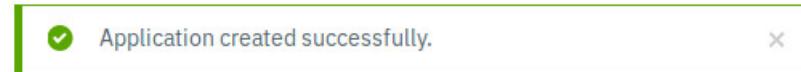
### Information

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.



## API Key and Secret

The API Key and Secret have been generated for your application.

### Key

010069709451404b2508e3b5cf187149  Show

### Secret

22c490ee5b007f8ad19a8fbae581e9b7  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. Clear any old values in the document.
- \_\_\_ e. Copy the values for both the API Key and secret from the application in the browser to the editor for the file.

```
*inventory-cred (~/) - gedit
Open Save
Client ID:
010069709451404b2508e3b5cf187149
Client Secret:
522c490ee5b007f8ad19a8fbae581e9b7
```

- \_\_\_ f. Save the changes to the file. Then, minimize the file.
- \_\_\_ g. Click **Continue** back in the browser of the Developer Portal.

- \_\_\_ h. 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 Think Application dashboard with the Subscriptions tab selected. The interface is divided into two main sections: **Credentials** and **Subscriptions**.

**Credentials:** This section contains fields for **Client ID** (displayed as a redacted string) and **Client Secret**. There is a **Verify** button and a  Show checkbox.

**Subscriptions:** This section displays a table with columns **PRODUCT** and **PLAN**. A message at the bottom states **No subscriptions found**, followed by a link [Why not browse the available APIs?](#).

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 IBM API Connect interface for the 'think-product' application. At the top, there's a green circular icon with a white cube symbol, followed by the text 'think-product' and '1.0.0' with a five-star rating. Below this, 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). In the 'Plans' section, there's a box labeled 'Default Plan' containing a blue 'Subscribe' button, which is outlined in red. Below the button is a 'View details' link.

- \_\_ f. On the Select Application page, click **Select app** in the Think Application window.

The screenshot shows a 'Select Application' window. It features a pink circular icon with a smartphone and gear symbol. Below the icon, the text 'Think Application' is displayed. Underneath that, the word 'Description' is followed by a brief description: 'A web application to browse items for sale, and to submit reviews.' At the bottom of the window is a large blue 'Select App' button, which is highlighted with a red rectangular border.

- \_\_\_ 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

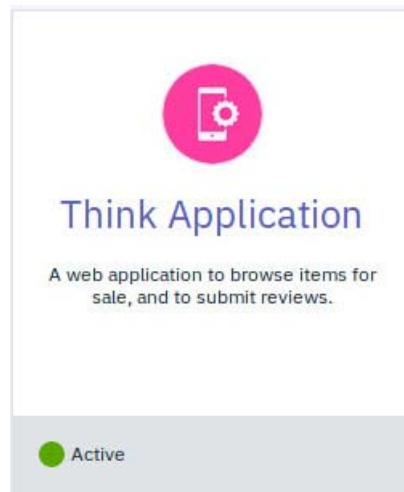
Think Application

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.

- \_\_\_ e. Click **Show** to show the Client ID.

## Think Application

Dashboard      Subscriptions

### Credentials

Credential for Think Application

Credential for Think Application

#### Client ID

010069709451404b2508e3b5cf187149

Show

#### Client Secret

Verify

- \_\_\_ f. 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 \*

.....

Cancel

Submit

Click **Submit**.

- \_\_\_ g. The client secret should verify successfully.



## Troubleshooting

If the validation fails, the client secret was not saved correctly.

 Validation of the credential failed. X

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, you 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. Click API Products in the top navigation bar
  - \_\_\_ b. Click the think-product 1.0.0 link



### 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.



- 
- \_\_\_ c.
  - \_\_\_ d. The think-product is open on the page.
  - \_\_\_ e. Click **financing** in the list of APIs.

The screenshot shows the 'think-product' API listing on the IBM Developer Portal. The product has a green star icon and a rating of 5 stars. It lists three APIs: 'financing 1.0.0', 'logistics 1.0.0', and 'inventory 1.0.0'.

| APIs                                                                                                |                                                                                                     |
|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
|  financing 1.0.0 |  logistics 1.0.0 |
|  inventory 1.0.0 |                                                                                                     |

- \_\_\_ f. 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's a search bar with a magnifying glass icon and the word "financing". To its right is a blue button with three vertical dots. Below the search bar, the API version "1.0.0" is shown along with a five-star rating. On the left, a sidebar has "Overview" selected. Under "Overview", there are links for "GET /calculate" and "Definitions". The main content area is titled "Overview". It includes a "Download" section with "Open API Document" links, and a "Type" section indicating "REST". Below that is an "Endpoint" section where "Production, Development:" is listed next to the URL "https://apigw.think.ibm/think/sandbox/financing". A "Security" section is present, showing "clientIdHeader" and the header "X-IBM-Client-Id" with the note "apiKey located in header".

- \_\_\_ g. Select GET /calculate from the left column.  
\_\_\_ h. 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.

- \_\_ i. Click the **Try it** tab in the right column.

## get.financingAmount

Details      **Try it**

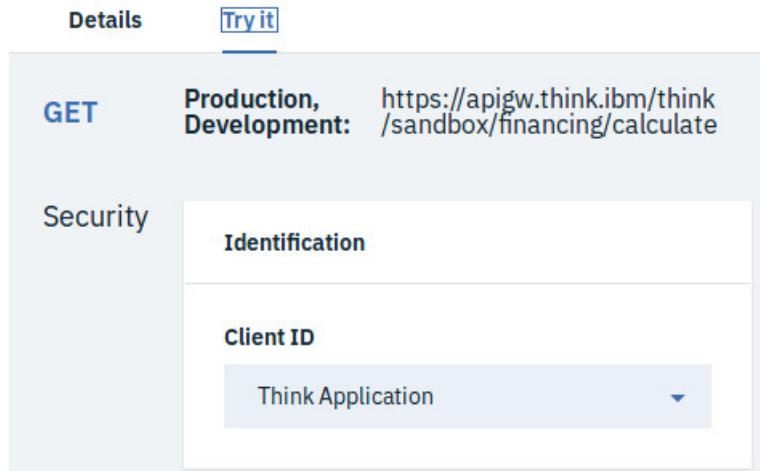
**GET**      **Production, Development:** <https://apigw.think.ibm/think/sandbox/financing/calculate>

Security

**Identification**

**Client ID**

Think Application ▾



The Developer Portal automatically inserts the client ID from the Think Application.

- \_\_ j. 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 test client interface for a REST API. 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 section titled "Query" is expanded, showing three parameters: "amount\*", "duration\*", and "rate\*". Each parameter has a text input field with a numeric value and up/down arrow buttons for adjustment. The "amount" field contains "300", the "duration" field contains "24", and the "rate" field contains "0.04". To the right of each input field is a "Generate" link. At the bottom of the screen are two large blue buttons: "Reset" on the left and "Send" on the right.

Click **Send**.

- \_\_ k. Confirm that the operation returned a status code of 200 OK.

### Response

```

Code: 200 OK
Headers:
content-type: application/json
x-ratelimit-limit: name=default,100;
x-ratelimit-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": {

```

- \_\_ l. Scroll down in the response area to see the calculated payment amount.

```

 "ser:paymentAmount": {
 "@xmlns": {
 "ser": "http://services.think.ibm",
 "soapenv": "http://schemas.xmlsoap.org/soap/envelope/"
 },
 "$": "12.51"
 }
 }

```



### Troubleshooting

If you do not receive a 200 OK response, this might be due to the two virtual machines being out of sync. Refer to the **Troubleshooting Issues** section in Appendix B.

```

Code: 401 Unauthorized
Headers:
content-type: application/json
{
 "httpCode": "401",
 "httpMessage": "Unauthorized",
 "moreInformation": "The API request is rejected because of no API Connect Gateway Service."
}

```



## Questions

How did the financing API return a successful response even though the inventory application is not running?

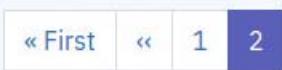
Answer: Although the financing API is part of the same think-product, the API operation is called independently from the inventory application. The financing API calls a SOAP service that runs on the Services domain of the DataPower Gateway. Essentially, the financing API uses a different back-end to the inventory API.

- 
- \_\_\_ 2. Test the operations from the **logistics** API definition.
    - \_\_\_ a. Click the API Products tab in the Developer Portal menu to display the products and APIs.
    - \_\_\_ b. Click the link in the think-product to display the list of APIs. The think-product is open on the page.
- 



## 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.



- 
- \_\_\_ c.
  - \_\_\_ d. Click **logistics** in the list of APIs.
- 

The screenshot shows the IBM Developer Portal interface. At the top, there's a header with a green circular icon containing a white cube, the text "think-product 1.0.0", and a five-star rating. Below the header, a section titled "APIs" lists three items:

- financing 1.0.0
- logistics 1.0.0
- inventory 1.0.0

- \_\_\_ e. The overview page for the logistics API is displayed.

The screenshot shows the API Management interface. At the top, there's a logo with a gear and the word "logistics" next to it, followed by "1.0.0" and a five-star rating. Below this is a navigation bar with "Filter" and a settings icon. The main area has a sidebar on the left with "Overview" selected, showing operations like "GET /shipping" and "GET /stores". The right side is the "Overview" panel. It displays the API name "logistics", version "1.0.0", and a "Download" button with a downward arrow. Below that are links for "Open API Document". The "Type" is listed as "REST". Under "Endpoint", it shows "Production, Development" and the URL "https://apigw.think.ibm/think/sandbox/logistics". The "Security" section includes a "clientID" field and an "X-IBM-Client-Id" field described as "apiKey located in header".

- \_\_\_ f. Select the GET /stores operation from the left column.  
\_\_\_ g. In the right column, click the Try it tab.

- \_\_ h. Type 15222 in the **zip** input parameter.

The screenshot shows a configuration interface for a REST API call. On the left, there's a sidebar with 'Parameter' and 'Definitions'. The main area has sections for 'Header' (Accept: application/json), 'Content-Type' (application/json), and 'Query'. Under 'Query', there's a field labeled 'zip\*' containing '15222'. At the bottom are 'Reset' and 'Send' buttons.

Click **Send**.

- \_\_ i. Confirm that the operation returns a status of 200 OK.

The screenshot shows the results of the API call. On the left, under 'Definitions', 'GET /stores' is selected. The 'Request' section shows the URL `GET https://apigw.think.ibm/think/sandbox/logistics/stores?zip=15222` and Headers: `Accept: application/json`, `X-IBM-Client-Id: 010069709451404b2508e3b5cf187149`. The 'Response' section shows the status code `Code: 200 OK` and 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](https://openstreetmap.org).

- 3. Sign out of the Developer Portal.
- 4. Close the web browser.

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 the 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.

# 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 6. 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/solutions/petstore_1.0.0.yaml</code>                                                                                                                        |
| <a href="#">Exercise 3, "Define an API that calls an existing SOAP service"</a>                | <code>~/lab_files/solutions/itemService.wsdl</code>                                                                                                                           |
| <a href="#">Exercise 4, "Create a LoopBack application"</a>                                    | <code>~/lab_files/loopback/complete</code><br><code>~/lab_files/solutions/notes_1.0.0.yaml</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/solutions/inventory_1.0.0.yaml</code><br><code>~/lab_files/solutions/financing_1.0.0.yaml</code><br><code>~/lab_files/solutions/logistics_1.0.0.yaml</code> |
| <a href="#">Exercise 8, "Declare an OAuth 2.0 provider and security requirement"</a>           | <code>~/lab_files/solutions/Inventory_oauth.yaml</code>                                                                                                                       |

Table 6. Model solution file for course exercises

| Exercise name                                                                                 | Completed exercise solution |
|-----------------------------------------------------------------------------------------------|-----------------------------|
| <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>       |
| <a href="#">Exercise 11, "Subscribe and test APIs"</a>                                        | <i>Not applicable</i>       |

---

# Appendix B. Troubleshooting Issues

This Appendix describes some issues that you might encounter when working through the exercises and troubleshooting tips for how to deal with them. Although issues might arise due to the two virtual machines getting out of sync, the student is encouraged to import a relevant solution file to verify the issue is with the environment, or the API configuration.

## IBM Remote Lab Platform - suspension of images

If the IRLP Skytap environment suspends the virtual machines, the DataPower and API Connect virtual machines might get out of synch. This can cause issues when testing the APIs in the Test client. Various error codes may be received as a result. This section provides troubleshooting tips in case this happens. The following status codes are covered:

- *400 Bad Request*
- *401 Unauthorized API Request*
- *401 Unauthorized: no API Connect Gateway Service*
- *401 Unauthorized: client id not registered*
- *404 Not found*
- *415 Unsupported Media Type*
- *500 URL Open error*
- *500 Internal Server Error*
- *503 API Error*
- *CORS Error*

## 400 Bad Request

**Issue:** When invoking an API, you get the following error:

**Response**

**Status code:**  
400 Bad Request

**Response time:**  
91ms

**Headers:**

```
apim-debug-trans-id: 426530149-Landlord-
apiconnect-c57d9fff-8290-465c-b4bf-65556c192fff
content-type: application/json
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,97;
```

**Body:**

```
{
 "code": 400,
 "type": "unknown",
 "message": "bad input"
}
```

**Solution:** Verify correct values are entered in the payload request.

## 401 Unauthorized API Request

**Issue:** If you leave the user signed on to the user interface for a significant amount of time, you may encounter the following error:



**Solution:** Close the browser and sign on again.

## 401 Unauthorized: no API Connect Gateway Service

**Issue:** When invoking an API, you get the following error:

| Response                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status code:<br>401 Unauthorized                                                                                                                                                            |
| Response time:<br>39ms                                                                                                                                                                      |
| Headers:<br>apim-debug-trans-id: 426530149-Landlord-apiconnect-4f5bf661-f803-4735-97af-65556c190ef6<br>content-type: application/json                                                       |
| Body:<br><pre>{<br/>  "httpCode": "401",<br/>  "httpMessage": "Unauthorized",<br/>  "moreInformation": "The API request is rejected because of no API Connect Gateway Service."<br/>}</pre> |

This error is due to the two virtual machines being out of synch.

**Solution:** Reset the environment following the [Reset Environment Procedures](#).

## 401 Unauthorized: Client id not registered

**Issue:** When invoking an API, you get the following error:

| Response                                                                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status code:<br>401 Unauthorized                                                                                                                     |
| Response time:<br>36ms                                                                                                                               |
| Headers:<br>apim-debug-trans-id: 426530149-Landlord-apiconnect-5ef29733-b9b1-4b8e-b767-65556c19f9ba<br>content-type: application/json                |
| Body:<br><pre>{<br/>    "httpCode": "401",<br/>    "httpMessage": "Unauthorized",<br/>    "moreInformation": "Client id not registered."<br/>}</pre> |

This occurs when an API with a client id has been published but the two virtual machines got out of sync.

**Solution:** To resolve this issue, republish the API to re-register the client id. It may take several minutes before the client id is successfully registered. If you get a **404 error**, try again

# 404 - Not Found

**Issue:** When invoking an API, you get the following error:

Response

---

Status code:  
404 Not Found

Response time:  
46ms

Headers:

```
apim-debug-trans-id: 426530149-Landlord-
apiconnect-0f4fcdae4-dfe2-47ff-bba2-65556c19d71f
content-type: application/json
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,95;
```

or you encounter a CORS error when attempting to invoke an API.

| Response                                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Status code:<br/>-1</p>                                                                                                                                            |
| <p>No response received. Causes include a lack of CORS support on the target server, the server being unavailable, or an untrusted certificate being encountered.</p> |

When you click the link in the CORS error, you receive a 404 error.

A screenshot of a Mozilla Firefox browser window. The title bar says "Not Found! - Mozilla Firefox". There are three tabs open: "API Connect", "Not Found!", and "(unknown)". The "Not Found!" tab is active. The address bar shows the URL "https://apigw.think.ibm/think/sandbox/v2/pet/1". Below the address bar are navigation icons (back, forward, search, etc.) and links for "Most Visited", "Getting Started", "Cloud Manager", "API Manager", and "API Developer Port". The main content area displays a large "404 - Not Found!" in bold black font, followed by the message "The requested URL was not found on this server".

This might occur for a few different reasons.

1. The endpoint was not located.

**Solution:** You should first retry the invocation. At times, although the endpoint is not located on the first try, it is found on subsequent tries.

2. The app-server value or target-url value in the API is incorrect.

**Solution:** Verify the app-server and target-url values are correctly configured. If they are, you may need to verify the Sandbox is attached (see below).

3. The Sandbox is not attached and the environments are out of sync.

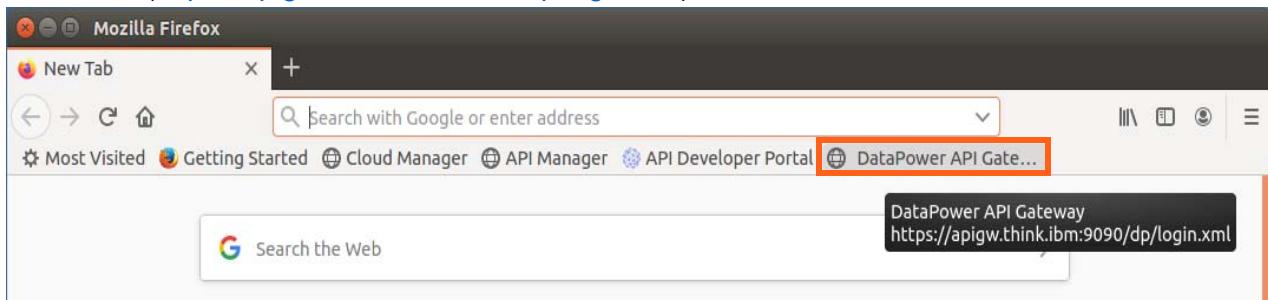
**Solution:** Follow these instructions:

- \_\_ 1. Log into the IBM DataPower API Gateway.

- \_\_ a. Click the **Firefox Web Browser** icon in the navigation bar on the left.



- \_\_ b. From the Firefox browser, click the **DataPower API Gateway** link (<https://apigw.think.ibm:9090/dp/login.xml>)



- \_\_ c. Enter the following credentials:

User: admin

Password: passw0rd

Domain: apiconnect

## IBM DataPower Gateway IDG.2018.4.1.1

IDG console at 10.0.0.20:9090

User name:

admin

Password:

.....

Domain:

apiconnect

Graphical Interface:

Blueprint Console

**Login**

Licensed Materials - Property of IBM Corp, IBM Corporation and other(s) 1999, 2018. IBM is a registered trademark of IBM Corporation, in the United States, other countries, or both.

- \_\_ d. Click **Login**.

\_\_ 2. Attach the Sandbox.

\_\_ a. Under the API collection section, click Add.

The screenshot shows the 'API Gateway: apiconnect' configuration page. On the left is a tree view of gateway components, with 'API Gateway apiconnect' selected. The main panel has a 'Status: up' indicator. In the 'Main' section, there are fields for 'Name' (set to 'apiconnect'), 'Enable administrative state' (checked), 'Comments' (empty), and 'Source protocol handler' (set to 'apiconnect\_https\_44'). Below these are sections for 'Crypto Identifiers' and 'SSL Server Profiles'. The 'API collection' section contains a dropdown menu with 'No items' and an 'Add' button, which is highlighted with a red box. At the bottom right are 'Apply' and 'Cancel' buttons.

\_\_ b. Verify the **think\_sandbox\_collection** is selected and click **Apply**.

**API Gateway: apiconnect**

Status: **up**

Name: apiconnect

Main

Enable administrative state:

Comments:

Source protocol handler: apiconnect\_https\_44

API collection: think\_sandbox\_collection

Share Rate Limit Count: yes

Apply Cancel

\_\_ c. Click **Save changes** to save changes to the configuration.

IBM DataPower Gateways | IDG console at 10.0.0.20:9090

apiconnect admin ? IBM

Services

All Services

| Service    | Status | Service Type | Front side URL        | Actions |
|------------|--------|--------------|-----------------------|---------|
| apiconnect | Up     | API Gateway  | https://10.0.0.20:443 |         |

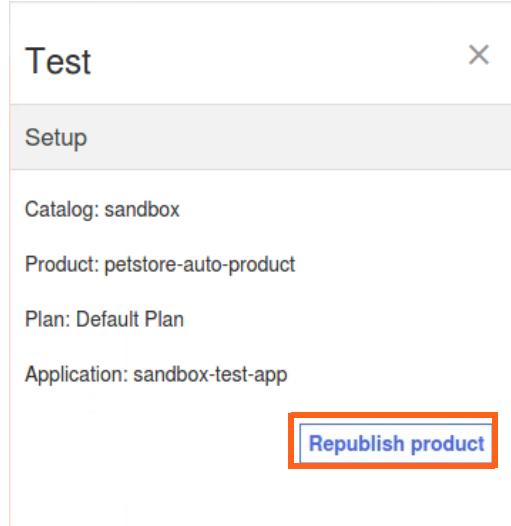
Total: 1

Save changes



## Troubleshooting

At times after a restart, the Sandbox does not appear. If this happens, republish an existing API to rebuild the Sandbox. When you return to the DataPower API Gateway console, you will see the option to save changes. Follow the procedures above to save the changes to the configuration.



## 415 Unsupported Media Type

**Issue:** When invoking an API, you get the following error.

| Response                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Status code:</b><br>415 Unsupported Media Type                                                                                                                                                                               |
| <b>Response time:</b><br>93ms                                                                                                                                                                                                   |
| <b>Headers:</b><br>apim-debug-trans-id: 426530149-Landlord-apiconnect-46452d1e-a030-4566-9928-65556c19896b<br>content-type: application/json<br>x-ratelimit-limit: name=default,100;<br>x-ratelimit-remaining: name=default,92; |

The HTTP 415 Unsupported Media Type error indicates that the server refuses to accept the request because the payload format is unsupported.

**Solution:** Verify the target-url is correct and that the URL in the invoke operation is correctly configured.

## 500 URL Open error

**Issue:** When invoking an API, you get the following error:

| Response                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Status code:<br/>500 URL Open error</p> <p>Response time:<br/>41ms</p> <p>Headers:</p> <pre>apim-debug-trans-id: 426530149-Landlord- apiconnect-52da6b73-a5b9-4bf5- b456-65556c193ad2 content-type: application/json x-ratelimit-limit: name=default,100; x-ratelimit-remaining: name=default,95;</pre> <p>Body:</p> <pre>{   "httpCode": "500",   "httpMessage": "URL Open error",   "moreInformation": "Could not connect to endpoint" }</pre> |

This occurs when the API endpoint cannot be reached.

**Solution:** Make sure whichever application the API is attempting to connect to is running. Verify the target-url is correctly configured.

## 500 Internal Server Error

**Issue:** When invoking an API, you get the following error:

### Response

#### Status code:

500 Internal Server Error

#### Response time:

543ms

#### Headers:

```
apim-debug-trans-id: 426530149-Landlord-
apiconnect-bdcaab0f-e81e-456d-
9908-65556c1960b9
content-type: application/json
x-global-transaction-id: 196c55655e2b1fb200000ec0
x-ratelimit-limit: name=default,100;
x-ratelimit-remaining: name=default,97;
```

#### Body:

```
{
 "httpCode": "500",
 "httpMessage": "Internal Server Error",
 "moreInformation": "Internal Error"
}
```

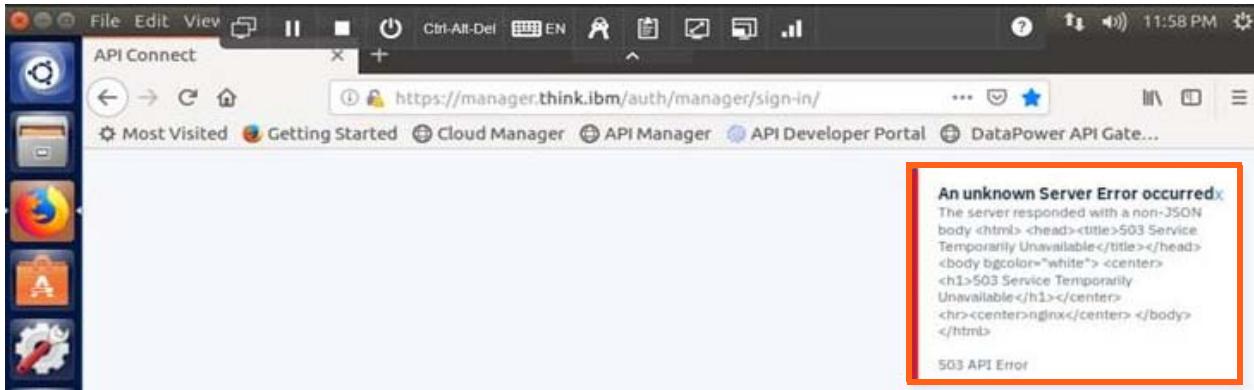
This occurs when a general error occurs when invoking the API. The most common causes are

- No value entered for required parameter.
- Incorrectly configured target-url.

**Solution:** Make sure the target-url is correctly configured and that values for all required parameters are provided.

## 503 API Error

**Issue:** When attempting to log into API Manager, you get the following error:



This error is due to the two virtual machines being out of sync or the environment not being completely started.

**Solution:** Reset the environment following the [Reset Environment Procedures](#).

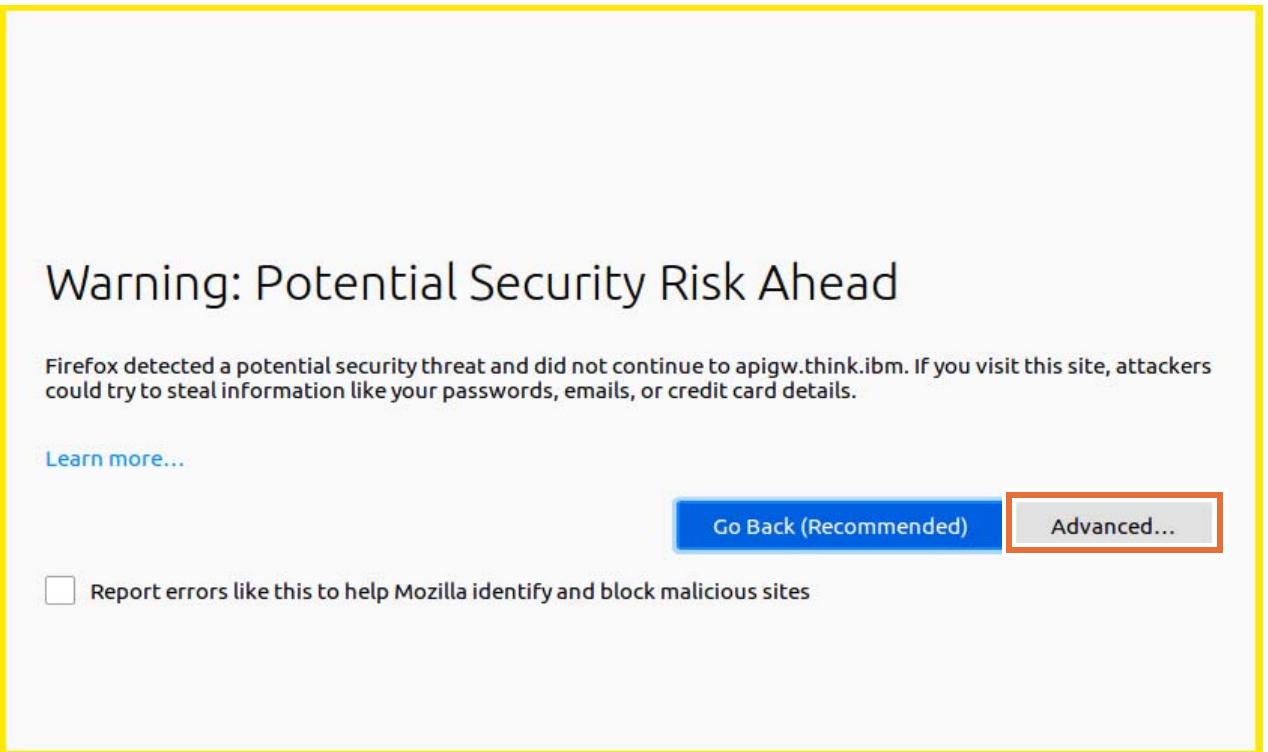
## CORS Error

**Issue:** When invoking an API, you get the following error:

| Response                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status code:<br>-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.<br><a href="https://apigw.think.ibm/think/sandbox/v2/pet/98798">https://apigw.think.ibm/think/sandbox/v2/pet/98798</a> |
| Response time:<br>37ms                                                                                                                                                                                                                                                                    |

- a. 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.

- \_\_\_ b. Click the link in the response area.
- \_\_\_ c. Accept the security exception from the web browser by clicking **Advanced...**



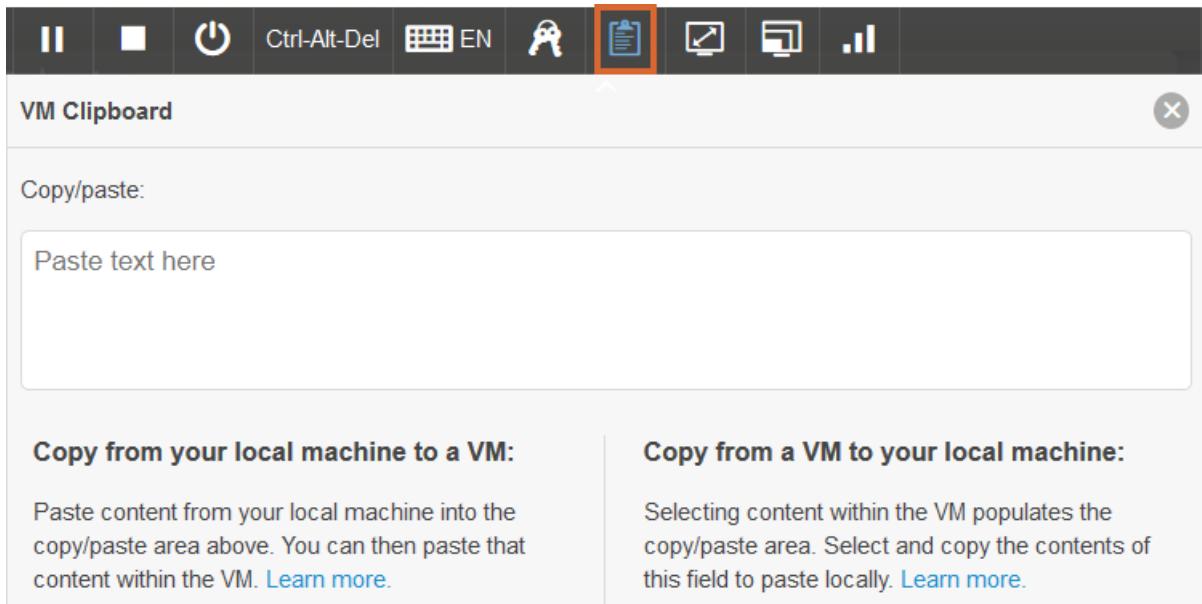
- \_\_\_ d. Click **Accept the Risk and Continue**.



- \_\_\_ e. Retry the invoke operation.

## IBM Remote Lab Platform - 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.

# 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

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.

| Pod Name                                          | Age  | Status    | Replicas | Ready | Reason |
|---------------------------------------------------|------|-----------|----------|-------|--------|
| r674f0bc86d-apiconnect-cc-repair-1549328400-dbs8z | 4d6h | Completed | 0/1      | 0/1   |        |
| r674f0bc86d-apiconnect-cc-repair-1549328400-llww4 | 3d2h | Error     | 0/1      | 0/1   |        |
| r674f0bc86d-apiconnect-cc-repair-1549328400-n9277 | 3d2h | Error     | 0/1      | 0/1   |        |
| r674f0bc86d-apiconnect-cc-repair-1549328400-pzk9t | 3d2h | Error     | 0/1      | 0/1   |        |
| r674f0bc86d-apiconnect-cc-repair-1549501200-v2phs | 31h  | Completed | 0/1      | 0/1   |        |
| r674f0bc86d-apiconnect-cc-repair-1549501200-zx7xw | 31h  | Error     | 0/1      | 0/1   |        |
| r674f0bc86d-apim-schema-init-job-ltcpz            | 9d   | Completed | 0/1      | 0/1   |        |
| r674f0bc86d-apim-v2-88674dd9f-g8r95               | 9d   | Running   | 1/1      | 1/1   |        |
| r674f0bc86d-client-dl-srv-b7fdf9767-6g4qz         | 9d   | Running   | 1/1      | 1/1   |        |

Then, from the terminal, type:

```
kubectl logs -n apiconnect [pod-name] > apim.logs
```

It might take a few minutes for the log file to be written to the current directory.

In the terminal, type:

```
tail -1000 apim.logs
```

The log file is displayed.

```
localuser@ubuntu:~$ kubectl logs -n apiconnect r674f0bc86d-apim-v2-88674dd9f-g8r95
> apim.logs
localuser@ubuntu:~$ tail -1000 apim.logs
 "updated_at": "2019-01-30T22:13:42.972Z",
 "url": "/api/cloud/registrations/038e6d68-aac1-47af-bb92-cc4f80ad2ca0"
}
Sun, 26 Jan 2020 03:41:09 GMT apim:routes:oauth2 [231801a80f778fd7285e6c8a3955c98a]

Sun, 26 Jan 2020 03:41:09 GMT apim:routes:oauth2 [231801a80f778fd7285e6c8a3955c98a]

Sun, 26 Jan 2020 03:41:09 GMT apim:routes:oauth2 [231801a80f778fd7285e6c8a3955c98a]
=====
Sun, 26 Jan 2020 03:41:09 GMT apim:routes:oauth2 [231801a80f778fd7285e6c8a3955c98a]
 \|/ Entering: oauth2::validateTokenContents
Sun, 26 Jan 2020 03:41:09 GMT apim:routes:oauth2 [231801a80f778fd7285e6c8a3955c98a]
 validatedToken: : {
 "jti": "87e87ad6-129a-44cb-b56a-106768a3293c",
 "namespace": "cloud",
 "aud": "/api/cloud/registrations/f5b6c3b4-34aa-43b0-aa74-a2f72fe3bccd",
 "sub": "/api/cloud/registrations/f5b6c3b4-34aa-43b0-aa74-a2f72fe3bccd",
 "iss": "IBM API Connect",
 "grant_type": "client_credentials",
 "exp": 1580038868,
 "iat": 1580010068,
```

