

Develop, publish, manage, and secure APIs with IBM API Connect V5.0

Student Exercise Guide

ERC 1.0



Table of Contents

[Lab 1 - Introduction to IBM API Connect](#)

[Lab 2 - Create a LoopBack Application](#)

[Lab 3 - Customize and Deploy an Application](#)

[Lab 4 - Configure and Secure an API](#)

[Lab 5 - Advanced API Assembly](#)

[Lab 6 - Working with API Products](#)

[Lab 7 - Consumer Experience](#)

Overview

This IBM API Connect (APIC) Proof of Technology (PoT) provides insight

to a complete solution which helps you:

- Quickly configure your API management solution and rapidly create new APIs.
- Manage your APIs with business-level controls by setting varying levels of consumption/entitlements and the ability to manage an application developer portal.
- Transform and grow your business with insights through detailed analytics with structured filtered searches and navigation capabilities.
- Acquire partners and innovative application developers, both internally and externally; through an attractive developer portal which can be tailored to your brand.
- Use leading standard industry practices to help secure, control, and optimize access to your APIs.
- Reduce resolution time with supplied operational monitoring and debugging capabilities and investments.

Introduction

To be a successful business and provide increased value to clients, employees, citizens, members or patients, one needs to continue to innovate while achieving cost optimization. To innovate while optimizing costs, you have an opportunity to expose key business services to business partners to drive new forms of collaboration, increase revenue opportunities, and provide higher value services to your customers. You can also expose your internal services to your internal developer communities, to drive application development consistency, reduce IT costs, and promote standard use of enterprise-approved services across your organization.

The expansion of the number of mobile devices, such as smart phones and tablets, as well as the growth of social media presents new and additional business opportunities to create new channels with customers and partners. To address these emerging opportunities, you require easy-to-use tools to define, assemble, secure, limit and monitor services, while providing a self-service experience for rapid developer on-boarding.

Business services are exposed with APIs. An API management solution is imperative to the success of externalizing the core services by providing easy assembly of new APIs, enabling security and different levels of service, providing management and insight to developers and business users, and socializing those APIs to developers, through communities and

portals. This enables organizations to participate in the API economy, with rapid, highly secure connections between API providers and API consumers.

This new release of IBM API Connect delivers a complete solution across the Create + Run + Secure + Manage set of capabilities that can be deployed on-premise in an enterprise's data center, or in the cloud. API Connect allows businesses as API providers to integrate the most important features that are necessary to enable the three most significant players in the API economy: application developers, business owners, and IT personnel.

API Connect provides capabilities for creating new applications and APIs, SDLC management, proxying, assembling, securing and scaling the APIs. This solution also provides detailed analytics and operational metrics to the business owner and a customizable developer portal to socialize the APIs and manage applications that can be used by the developers. The developer portal enables self-service registration and provides links to social communities.

Requirements

To complete the labs in this workbook, you'll need the following:

- A network attached workstation with sufficient memory (16GB min).
- VMware Workstation v9 for Windows or VMWare Fusion v8 for OSX to run the supplied student VMware images.
- Access to the API Connect POT Stack – which includes
- IBM DataPower Gateway Appliance (Physical or virtual) with Firmware 7.5.0.0 or greater
- IBM API Management Hypervisor, version 5 or higher.
- Developer Portal, version 5.0.0.0 or higher
- Primary host OS VM for the POT running on Xubuntu Linux

Student workstation username and password

The WU500 lab environment provides the *API Connect POT Stack* as a set of four virtual machines. To complete the lab exercises, open the Xubuntu Linux virtual machine with the following username and password:

- Username: `student`
- Password: `Passw0rd!`

Guidance

The following symbols appear in lab guides at places where additional guidance is available.

| Icon | Purpose | Explanation |
|-------------------------------------------------------------------------------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Important! | This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive. |
|  | Information | This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know. |
|  | Troubleshooting | This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information. |

Proceed to [Lab 1 - Introduction to IBM API Connect](#)

Lab 1 - Introduction to IBM API Connect

In this lab, you'll gain a high level understanding of the architecture, features, and development concepts related to the IBM API Connect (APIC) solution. Throughout the lab, you'll get a chance to use the APIC command line interface for creating LoopBack applications, the intuitive Web-based user interface, and explore the various aspects associated with solution's configuration of RESTful based services as well as their operation.

Lab 1 - Objective

In the following lab, you will learn:

- How to create a simple LoopBack application
- How to create a Representational State Transfer (REST) API definition
- How to test a REST API
- How to publish an API to the Developer Portal

Lab 1 - Case Study Used in This Tutorial

In this tutorial, you will create the default Hello World LoopBack application. We will look at the created artifacts to get a better understanding of what is created for you. The hello-world application is a simple RESTful service that holds a set of "notes" in memory. In this lab we may be creating a new RESTful service and API, but in future labs you'll be creating APIs with existing services.

Lab 1 - Before You Begin

For this lab, you will be using a VMWare image running Xubuntu that is pre-installed with the API Connect Developer Toolkit. Ensure the image is up and running before starting. If you have any questions, please consult your lab instructor.

Lab 1 - Student workstation username and

password

To start the lab exercises, open the Xubuntu Linux virtual machine with the following username and password:

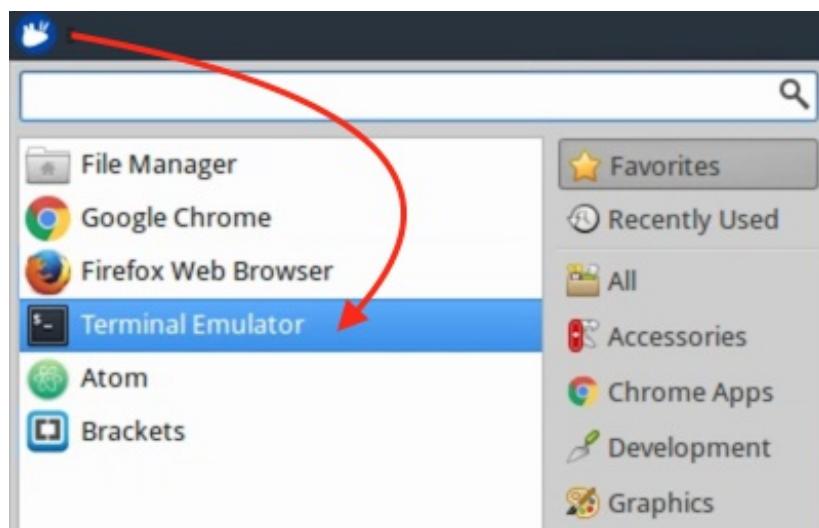
- Username: student
- Password: Passw0rd!

Lab 1 - Step by Step Lab Instructions

1.1 - Creating a hello-world Application

You will use the API Connect Developer Toolkit command line interface to create the initial application and explore the created artifacts.

1. Open the terminal emulator.



2. From the terminal command line type:

```
apic loopback hello-world
```

This command starts the application generator, Yeoman, to help scaffold the new project. Just press enter for each of the three questions.

```
  _-----_
 |           |
 |(o)--|   | Let's create a LoopBack |
 |       |     application!      |
 |-----|
 |  _`U`_  |
 | /__A__\ |
 | | ~  |
 | ._. |
 | |°  Y `|
 |
 ? What's the name of your application? hello-world
 ? Enter name of the directory to contain the project: hello-world
     create hello-world/
     info change the working directory to hello-world

? What kind of application do you have in mind? hello-world (A project containing
     a basic working example, including a memory database)
```

This creates an application named "hello-world" in a directory of the same name. The application is a basic Hello World application. You will see a lot of messages printed to the command line window. It is creating a few resources for you and installing the various node modules. Once the node modules are loaded you'll notice that the process creates swagger and product definitions for you. Finally, the process displays some hints about what to do next. Since we've been given such lovely suggestions about what to do next, we may as well follow the first one at least.

3. Change directories to the project directory:

```
cd hello-world
```

4. List the directory:

```
$ ls
client  common  definitions  node_modules  package.json
server
```

If you're familiar with LoopBack or Node.js applications, some of the

directories may be familiar to you. If not, here is a description of some of what is created.

| Directory or file | Description |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| client | If the application has a front end, this is where the HTML, CSS, JavaScript, etc would be located. For our sample application there is only a stubbed out README.md. |
| common | Files common to both the server and the client application. |
| common/models | This sub-directory contains all the model JSON and JavaScript files. By model we're referring to a data model. |
| common/models/note.js | Custom script for the note data model. This file contains the implementation of the methods defined by the model definition file. |
| common/models/note.json | The note model definition file. This file contains the definition of the properties and methods for this model. |
| definitions | This directory contains the definitions for the APIs and the product contained in this application. |

| | |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| definitions/hello-world-product.yaml | YAML file for the the hello-world product. Which includes default plans for testing locally. |
| definitions/hello-world.yaml | Swagger definition file for the hello-world API. Includes information about the REST paths and operations, schemas for data models, security requirements, etc. |
| node_modules | Directory containing all the required node modules for the default application. |
| package.json | Standard Node.js package specification. Most importantly it contains the application package dependencies. |
| server | This directory contains the Node.js application and configuration files. We'll not look at them all in this tutorial, but here are a few. |
| server/server.js | The main application script for this application. Note: this is defined in the package.json file. |
| server/config.json | This file contains the global application settings, such as REST API root, hostname and port. |

| | |
|--------------------------|----------------------------------------------------------------------------------------------------------------|
| server/model-config.json | This file binds models to data sources and specifies whether a model is exposed over REST, among other things. |
| server/datasources.json | This file is the data source configuration file. |



*.md files, such as that found in the client directory, are markdown files used for internal documentation.

1.2 - Launching the hello-world Application

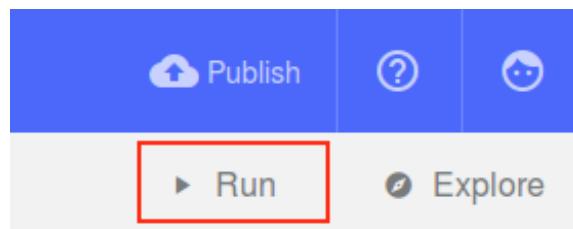
1. Now that we've explored what is created by the application generator, let's move on to the API Designer. From the command line:

```
apic edit
```

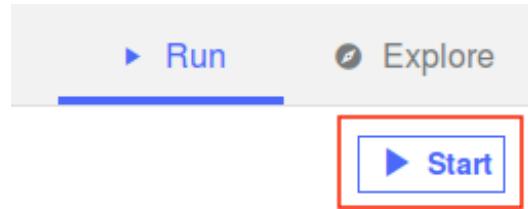
Alternatively, if you'd rather not login to bluemix:

```
SKIP_LOGIN=true apic edit
```

2. To test our hello-world services, click on the run button to open the application launcher.



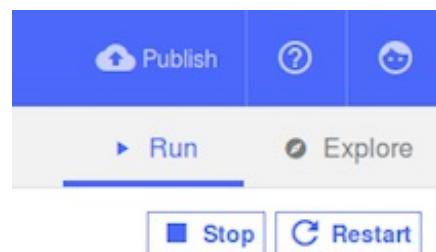
3. Next, click on the start button to launch the hello-world application.



- Once start completes, you should see a screen similar to this:

Running
Application: <http://127.0.0.1:4001/>
Micro Gateway: <https://127.0.0.1:4002/>

- Notice that once the application is up and running, stop and restart buttons will appear on the right side of the screen:



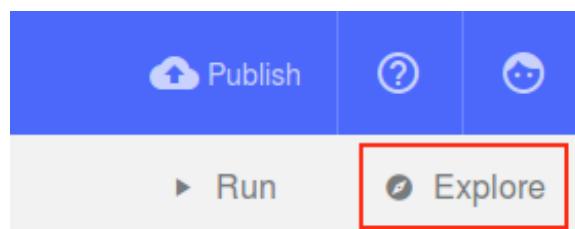
At this point we're ready to Explore and test our services.



We used the web-based editor to launch the application. There's also a command provided with the API Connect Developer Toolkit that can be utilized from the terminal to launch the application: `apic start`

1.3 - Testing the `hello-world` Application

- Click the `Explore` button to switch to the API Explorer view.



You will see all the exposed service paths displayed.

- Now we're going to test the services using the GUI presented on the explore screen. You'll notice that on the left several REST services are defined for us. In particular, take a look `POST /notes` and `GET /notes`.

If you're not familiar with GET and POST, they are HTTP methods (sometimes called verbs). The POST method is used for creation calls to the service. The GET method is used to retrieve information from a service. In this case, we see that both methods are used against the `/notes` path. So, POST will create a `note` and GET will retrieve all the `notes` that have been created.

- Start by creating a couple of notes. Click the `POST /notes` link in the left column. The other columns will scroll so that you're looking at information and controls pertaining to the `POST /notes` service.
- To test creation of a note, scroll down in the right hand column until the `Call operation` button is visible at the bottom. Just above the `Call operation` button you'll see a `Generate` link. This link will generate dummy data for you to create a test call to this service.
- Press the `Generate` link to generate some sample data.

Parameters

```
Model instance data  
data  
{  
    "title": "voluptatem",  
    "content": "laboriosam ipsa qui",  
    "id": -10426196.489835344  
}
```

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

Your data will look different, but you're ready to test the service.

6. Go ahead and press the `Call operation` button and scroll down to the `Response` section to see the results.

In the results, you should see a `Code: 200 OK` which indicates that a new `note` was created. If you received a different response, see the troubleshooting steps below.

Response

```
Code: 200 OK  
Headers:  
content-type: application/json; charset=utf-8  
x-ratelimit-reset: 3561141  
x-ratelimit-limit: 100  
x-ratelimit-remaining: 99  
  
{  
    "title": "nostrum in cum necessitatibus sapiente",  
    "id": 1  
}
```



You may see an error displayed that mentions a CORS issue. This has to do with certificates in your browser. Go ahead and click the given link to rectify this, accept any certificate, close the opened tab, and press the `Call operation` button again. Additionally, be sure not to skip step 5, as doing a `POST` operation without generating payload will cause an error.

Response

Code: -1

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

Clicking the link below will open the server in a new tab. If the browser displays a certificate issue, you may choose to accept it and return here to test again.

<https://localhost:4002/api/notes>



If you see a 500 error like the one below, make sure you press the `Generate` button before you press the `Call operation` button again. Otherwise, you're trying to create a duplicate note.

Response

Code: 500 Internal Server Error

Headers:

content-type: application/json; charset=utf-8

x-ratelimit-reset: 3594231

x-ratelimit-limit: 100

x-ratelimit-remaining: 98

{

 "error": {

 "name": "Error",

 "status": 500,

 "message": "Duplicate entry for note.id",

 "stack": "Error: Duplicate entry for note.id\nat Memory._createSync (/Users/tcat/Documents/data/gigs

- Once you have created one note, go ahead and create another one or two.



It should be noted that you don't need to use the `Generate` link. You can type data directly into the `Parameters`. You can also use `Generate` to create a template for you to use and then change the generated parameters. You may also notice that not all the parameters are always generated. This is because only the `title` parameter is required. Try pressing `Generate` several times to get a feel for how it works.

- Finally, let's test the `GET /notes` service. We should have two,

three, or more notes created at this point. In the left hand column click the `GET /notes` link.

```
hello-world 1.0.0
POST /notes
PUT /notes
GET /notes
GET /notes/{id}/exists
HEAD /notes/{id}
```

9. Scroll down to the `Call operation` button and press it. Then scroll down to the results.

Response

```
Code: 200 OK
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-reset: 3571999
x-ratelimit-limit: 100
x-ratelimit-remaining: 95

[
  {
    "title": "test test test",
    "id": 1
  },
  {
    "title": "quia itaque est minima",
    "content": "molestiae iure nostrum magnam",
    "id": 2
  }
]
```

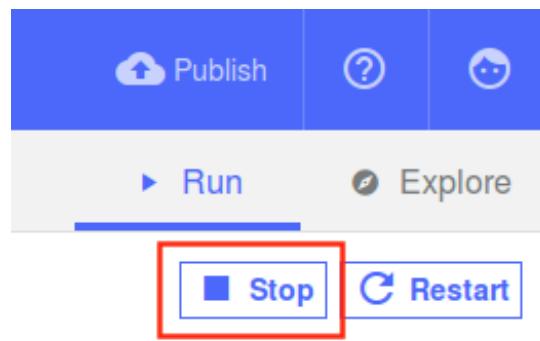
You should see all the notes that you generated in the result set.



If you see an empty array, `[]`, as your result, then you've not successfully created any notes. This is also true if you stop the application and restart it. With the hello-world example, we're using an in-memory database which means that nothing is persisted to disk. So, it is lost when the server is stopped and restarted. Lab 2 will walk through how to connect to your application to a persistent data source.

10. At this point, we are done testing the app locally. Click on the `Run` button again to return to the application launch screen.

11. Click on the `Stop` button to stop the application.



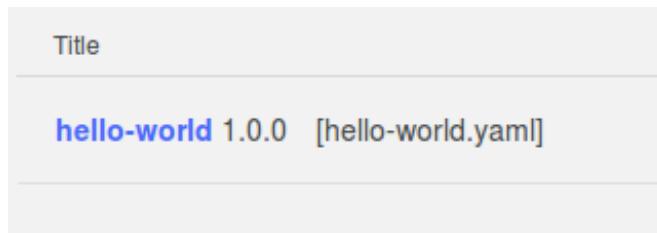
1.4 - Prepare the API for Publishing

In the previous steps, you started the new application along with a MicroGateway locally on your developer work station. Our API Connect infrastructure uses DataPower as the gateway though, and not the MicroGateway. In order to publish the API definition to the API Connect servers, we'll need to change the Gateway Policies to utilize DataPower.

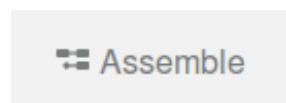
1. Click on the `APIs` option from the menu bar.



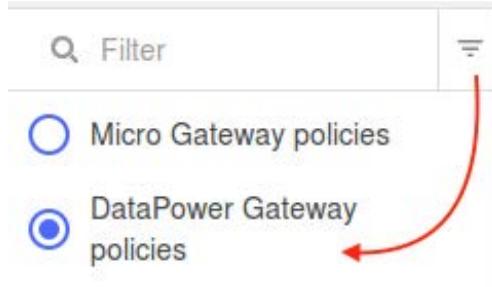
2. Click on the `hello-world` API from the list.



3. Click on the `Assemble` option from the API menu bar.



4. Click on the filter arrow to display the option to select the set of gateway policies and choose the option for `DataPower Gateway Policies`.



5. Click on the Save icon to save the API.



1.5 - Publishing the API to the Developer Portal

1. Click the `Publish` icon.



2. Select `Add and Manage Targets` from the menu.
3. Select `Add a different target`.



4. Provide connection information to sign into the IBM API Connect management server, then click the `Sign in` button:

- API Connect host address: `mgr.think.ibm`
- Username: `student@think.ibm`
- Password: `Passw0rd!`

Sign in to IBM API Connect

API Connect host address
mgr.think.ibm

Username
student@think.ibm

Password

[Back](#) [Cancel](#) [Sign In](#)

5. On the "Select an organization and catalog" screen, choose the **Sandbox** catalog and click the **Next** button.

Select an organization and catalog

Organization
Sales ▾

Search

None

Sandbox

6. On the "Select an App" screen, choose **None** application and click the **Save** button.

Our offline toolkit environment is now configured to speak to the central management server.

7. Click **Publish** button once more and select our target, indicated by the grey highlighting:

Other Orgs

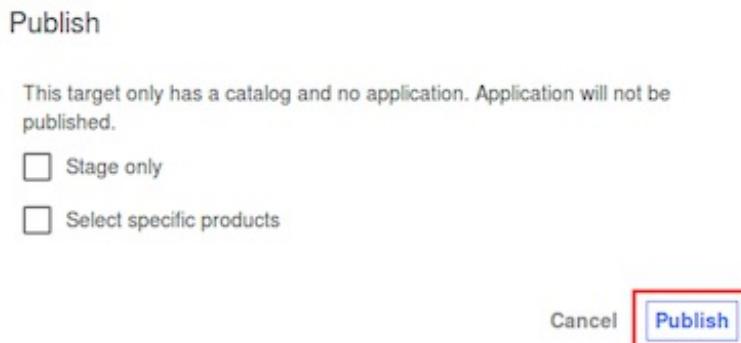
Catalog: Sandbox (sb)
Org: Sales (sales)
Server: mgr.think.ibm

Add and Manage Targets

8. Here we have the opportunity to select what gets published. If we were working on multiple API products as part of this project, we could choose specific ones to publish.

Also, the option exists to only Stage the product. A Stage-only action implies that we'll push the configuration to the Management server, but not actually make it available for consumption yet. The reason for doing this may be because your user permissions only allow staging, or that a different group is in charge of publishing Products.

Click the `Publish` button:

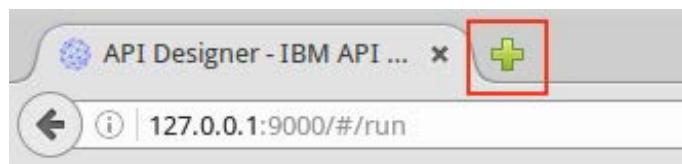


9. Click the `Publish` button to make the API available in the Developer Portal.
10. Wait a moment while the Product is published, a Success message will appear letting you know the step is complete:



1.6 - Browsing the API in the Developer Portal

1. Open a new tab in the Firefox web browser by clicking on the new tab `+` button.



2. A bookmark is already saved for the `Portal`, click on the bookmark to open the portal home page.

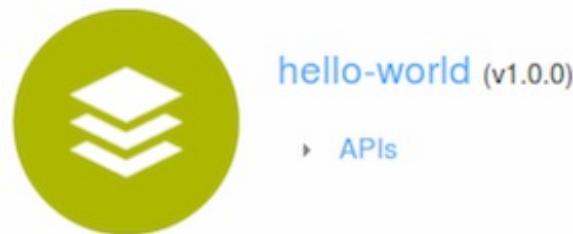


3. Click on the `API Products` link to see the available products published to the portal.



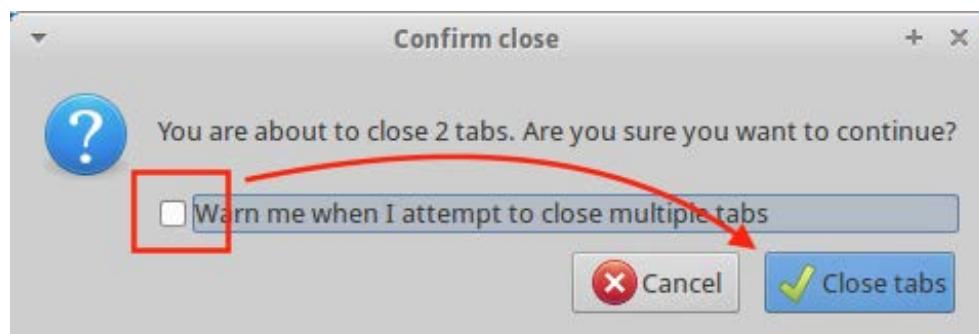
4. You should see the published `hello-world` API in the list of

products.



It should be noted that the hello-world application has more to it than we've shown in this lab. You're encouraged to dig in and discover the custom method that's implemented. In future labs, we'll be doing more work in the Developer Portal. For instance we'll be customizing the portal theme, registering an application, subscribing to APIs and testing them from a separate consumer application.

1. Close the `Firefox` web browser. If a warning is presented about closing multiple tabs, deselect the option to notify you in the future and click the `Close Tabs` button.



2. In the terminal, use the `control+c` keyboard command to quit the API Designer program.

Lab 1 - Conclusion

Congratulations! You have developed and published your first API!

In this lab you learned:

- How to create a simple LoopBack application
- How to create a Representational State Transfer (REST) API definition
- How to test a REST API
- How to publish an API to the Developer Portal

Proceed to [Lab 2 - Create a LoopBack Application](#)

Lab 2 - Create a Loopback Application

In this lab, you'll gain a high level understanding of the architecture, features and development concepts related to the IBM API Connect (APIC) solution. Throughout the lab, you'll get a chance to use the APIC command line interface for creating LoopBack applications, the intuitive web-based user interface, and explore the various aspects associated with the solution's configuration of RESTful based services as well as their operation.

At the end of this lab you will have created an new application which provides access to product inventory via a set of API resource operations.

Lab 2 - Objective

In the following lab, you will learn:

- How to create a multi-model Loopback application
- How to create a Representational State Transfer (REST) API definition using IBM Connect API Designer
- How to create a Representational State Transfer (REST) API definition using IBM Connect Command Line
- How to use the Loopback MySQL Connector
- How to use the Loopback MongoDB Connector
- How to test a REST API
- How to create relationships between models

Lab 2 - Case Study Used in this Tutorial

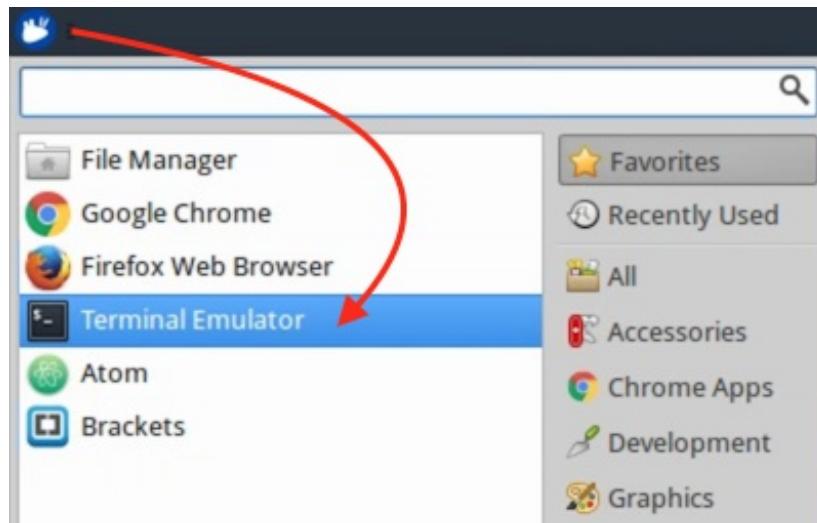
ThinkIBM is a company which sells historical and rare IBM machinery.

ThinkIBM wants to create easier access to their inventory database through a collection of APIs. Additionally, the application should also support the ability for buyers to leave reviews. As an application developer, you will create the application that provides access to product inventory.

Lab 2 - Step by Step Lab Instructions

2.1 - Create a Working Directory

1. Return to the `Terminal Emulator` by selecting it from the task bar, or if you closed it previously you can re-launch it from the favorites menu.



2. Create a project directory in the student's home folder called ThinkIBM. In the terminal, type:

```
mkdir ~/ThinkIBM
```

3. Change to the new `ThinkIBM` directory by typing:

```
cd ~/ThinkIBM
```

2.2 - Create the Inventory App

To create your Inventory Application you will use LoopBack technology that comes with the API Connect Developer Toolkit. LoopBack enables you to quickly compose scalable APIs, runs on top of the Express web framework and conforms to the Swagger 2.0 specification. LoopBack is a highly-extensible, open-source Node.js framework that enables you to:

- Create dynamic end-to-end REST APIs with little or no coding.
- Access data from Relational and NoSQL Databases, SOAP and other REST APIs.

- Incorporates model relationships and access controls for complex APIs.

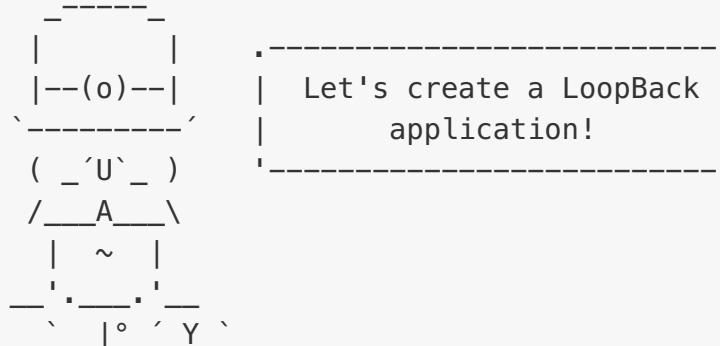
LoopBack consists of:

- A library of Node.js modules.
- Yeoman generators for scaffolding applications.

1. From the command line terminal, type the following command to create the `inventory` application:

```
apic loopback inventory
```

2. You will be asked to name your application.



```
? What's the name of your application? (inventory)
```

Since you already supplied the name of the application as part of the previous step you can keep the default by pressing the `Enter` or `Return` key.

3. Next you will be asked to supply the name of the directory where the application will be created.

LoopBack will default the project directory name to the name of the application.

Press the `Enter` or `Return` key to accept the default value of `inventory`.

1. Next you will be asked to select the type of application.

Use the arrow keys to select the `empty-server` option and press the `Enter` or `Return` key.

```
> empty-server (An empty LoopBack API, without any configured models or datasources)
```

- At this point, the project builder will install the core dependencies for our Node.js application.

Please wait until you see the `Next steps:` section.

- Change to the newly created `inventory` directory:

```
cd inventory
```

2.3 - Create a Data Source Connector to MySQL

The datasource is what allows the API to communicate with the backend data repository. In this case we will be using MySQL to store the inventory item information.

There are two parts to this. First is the definition of how to connect to the backend system. They second is downloading the actual loopback connector for MySQL. The connector is akin to an ODBC or JDBC connector.

- In your terminal ensure that you are in the `~/ThinkIBM/inventory` directory. You prompt should be

```
student@xubuntu-vm:~/ThinkIBM/inventory .
```

Optionally, you can type `pwd` in your terminal and it should return `/home/student/ThinkIBM/inventory`.

- In your terminal, type:

```
apic create --type datasource
```

The terminal will bring up the configuration wizard for our new datasource. The configuration wizard will prompt you with a series of questions. Some questions require text input, others offer a selectable menu of pre-defined choices.

Answer the questions with the following data:

```
? Enter the data-source name: mysql-connection
? Select the connector for mysql-connection:
  > MySQL (supported by StrongLoop)
? host: mysql.think.ibm
? port: 3306
? user: student
? password: Passw0rd!
? database: think
? Install loopback-connector-mysql@^2.2 (Y/n) Y
```



By typing Y (Yes) to the question

Install loopback-connector-mysql@^2.2 , the MySQL Connector will be downloaded and saved to your project automatically. This will create a connection profile in the ~/Thin kIBM/inventory/server/datasources.json file. It is effectively the same as running the following to install the connector:

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

For more information on the LoopBack Connector for MySQL, see:

<https://www.npmjs.com/package/loopback-connector-mysql>

2.4 - Launch the API Connect Designer

1. Ensure you're in the ~/ThinkIBM/inventory directory, then type the following command:

```
apic edit
```

The Firefox web browser will launch and automatically load the designer screen.

2. Now that the API Designer is running, you should see the start page with your inventory API.

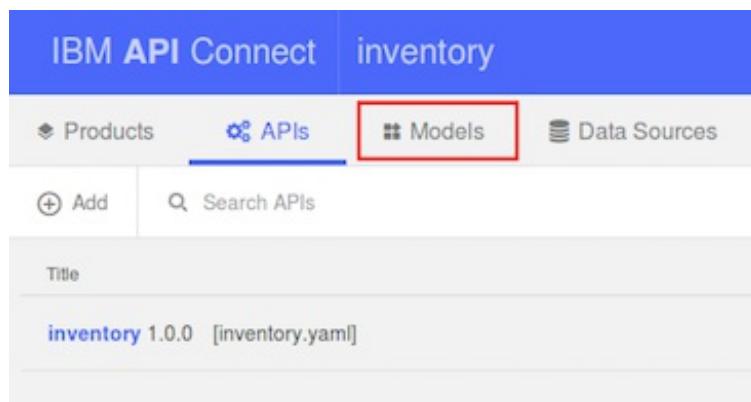


This API was created as a result of the generation of our LoopBack application.

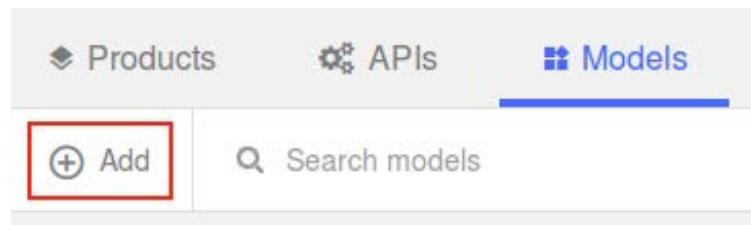
2.5 - Create a Model for the Inventory Items

In this section, you will define the `item` data model for our `inventory` API and attach it to the MySQL data source. LoopBack is a data model driven framework. The properties of the data model will become the JSON elements of the API request and response payloads.

1. Click the `Models` tab.



2. Click the `+ Add` button.



3. In the New LoopBack Model dialog, enter `item` as the model name, then click the `New` button.

New LoopBack Model

Name

Cancel

4. When the Model edit page for the item model displays, select the `my sql-connection` Data Source:

| | |
|--------------------------------------------|------------------|
| Name | item |
| Plural | |
| Base Model | PersistedModel |
| Data Source | mysql-connection |
| <input checked="" type="checkbox"/> Public | |

2.6 - Create Properties for the item Model

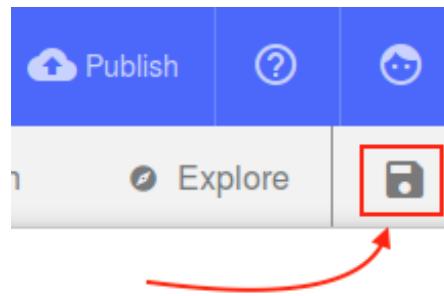
The item table in the MySQL database has 6 columns that will need to be mapped as well. To start creating properties for the item model:

1. Click the + button in the **Properties** section.
2. The item data model consists of six properties. Use the data below to add each of the properties:

| |
|-------------------------------------|
| Required: yes |
| Property Name: name |
| Type: string |
| Description: item name |
| ----- |
| Required: yes |
| Property Name: description |
| Type: string |
| Description: item description |
| ----- |
| Required: yes |
| Property Name: img |
| Type: string |
| Description: location of item image |

```
    | Required: yes
    | Property Name: img_alt
    | Type: string
    | Description: item image title
    |
    | Required: yes
    | Property Name: price
    | Type: number
    | Description: item price
    |
    | Required: no
    | Property Name: rating
    | Type: number
    | Description: item rating
```

3. Scroll to the top of the page and click the `Save` button to save the data model.

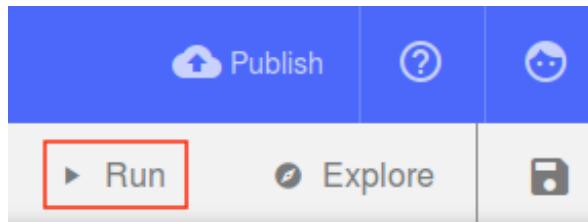


4. Click the `All Models` link to return to the main API Designer page.

2.7 - Verify API

To confirm that the API has been correctly mapped and can interface with the MySQL datasource, you will run the server and test the API.

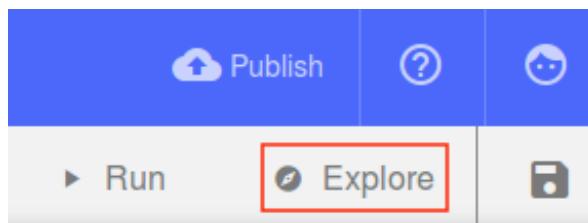
1. Click the `Run` button to open the Run page.



2. Click the `Start` button to start the `inventory` LoopBack application. In addition to the application, a MicroGateway will also be started.
3. Wait a moment while the servers are started. Proceed to the next step when you see the following:

Running
Application: <http://127.0.0.1:4001/>
Micro Gateway: <https://127.0.0.1:4002/>

4. Click the `Explore` button to review your APIs.



5. On the left side of the page, notice the list of paths for the `inventory` API. These are the paths and operations that were automatically created for you by the LoopBack framework simply by adding the `item` data model. The operations allow users the ability to create, update, delete and query the data model from the connected (MySQL) data source.
6. Click the `GET /items` operation.

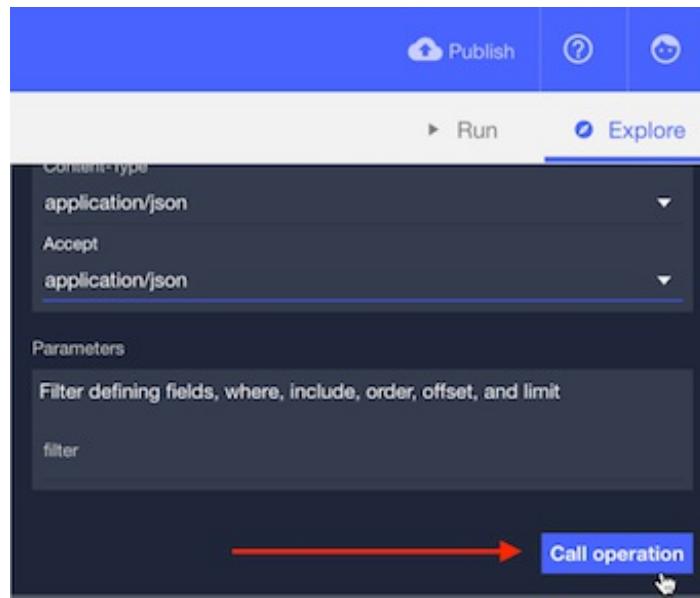


7. By clicking the `GET /items` operation, your screen will auto-focus to the correct location in the window. In the center pane you will see a summary of the operation, as well as optional parameters and responses.

On the right side you will see sample code for executing the API in various programming languages and tools such as cURL, Ruby, Python, PHP, Java, Node, Go, and Swift.

In addition to the sample code, if you look further down the page you will see an example response, URL, API identification information, and API parameters.

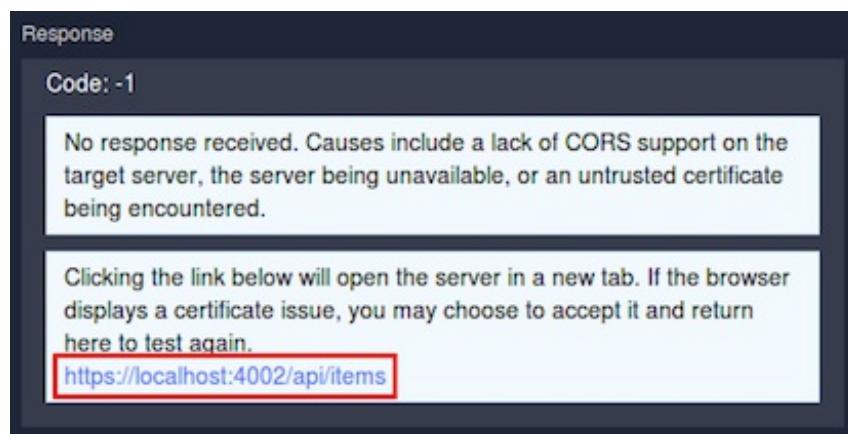
8. Scroll down slowly to locate the `Call operation` button.



9. Click the `Call operation` button to invoke the API.



The first time you invoke the API, you may receive an error. The error occurs because the browser does not trust the self-signed certificate from the MicroGateway. To resolve the error, click on the link in the response window and accept the certificate warning.



10. Once complete, return to the API explorer and click on the `Call ope`

ration button again.

11. Scroll down to see the Request and Response headers.

```
Request
GET https://localhost:4002/inventory/items
APIm-Debug: true
Content-Type: application/json
Accept: application/json
X-IBM-Client-Id: default
X-IBM-Client-Secret: SECRET
```

```
Response
Code: 200 OK
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-limit: 100
x-ratelimit-remaining: 99
x-ratelimit-reset: 3599999
```

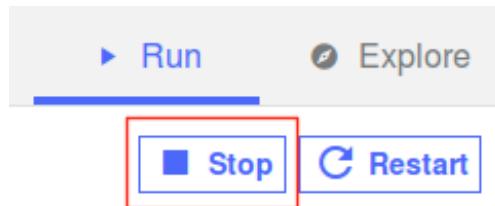
12. Scroll further and the payload returned from the GET request is displayed.

```
[
  {
    "name": "Dayton Meat Chopper",
    "description": "Punched-card tabulating machines and time clocks...",
    "img": "images/items/meat-chopper.jpg",
    "img_alt": "Dayton Meat Chopper",
    "price": 4599.99,
    "rating": 0,
    "id": 5
  },
  ...
]
```

13. Test the `GET /items/count` operation by following the same process above. You should receive a count of 12 inventory items.

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

14. Return to the `Run` screen and click the `Stop` button to stop the Inventory application and MicroGateway.



2.8 - Create a MongoDB Data Source

So far, we have created a LoopBack application which provides APIs around our inventory items stored in a MySQL database.

In the next section, you will create the data model for item reviews which will use MongoDB to store the review data.

First you must create a data source entry for the MongoDB.

1. Click the `x` button on the Firefox tab or window to close the browser.
2. Select the `Terminal Emulator` from the taskbar to open the command line.
3. Even though we closed the browser, the API Designer application itself is still running.

Hold the `control` key and press the `c` key to end the API Designer session:

```
control+c
```

This will return you to the command line prompt.

4. Type the following command to create a data source for MongoDB:

```
apic create --type datasource
```

5. The terminal will bring up the configuration wizard for our new datasource. The configuration wizard will prompt you with a series of questions. Some questions require text input, others offer a selectable menu of pre-defined choices.

Answer the questions with the following data:

```
? Enter the data-source name: mongodb-connection
? Select the connector for mongodb-connection:
  > MongoDB (supported by StrongLoop)
? host: mongo.think.ibm
? port: 27017
? user:
? password:
? database: think
? Install loopback-connector-mongodb@^1.4 (Y/n) Y
```



By typing Y (Yes) to the question Install loopback-connector-mongo@^1.4, the MongoDB Connector will be downloaded and saved to your project automatically. This will create a connection profile in the

`~/ThinkIBM/inventory/server/datasources.json` file.

It is effectively the same as running the following to install the connector

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

For more information on the LoopBack Connector for MongoDB, see:

<https://www.npmjs.com/package/loopback-connector-mongodb>

2.9 - Create Model for Reviews

The `review` data model will be used to store item reviews left by buyers. The reviews will be stored in a MongoDB.

In the earlier steps, you used the API Designer User Experience to create

a data model. This time you will use the command line to create the `review` model.

1. Type the following command to create the `review` data model:

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

2. Enter the properties for the `review` model:



You will **not** expose the review mode as a REST API. This is because you create a relationship between item and review later that will create the REST APIs you will use.

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

3. Continue using the wizard to add properties for the `review` model:

4. The first property is the `date` property:

```
Enter an empty property name when done.
? Property name: date
? Property type:
  > date
?Required? Y
?Default value [leave blank for none]: <leave blank>
```

5. Next add the `reviewer_name` property:

```
Let's add another review property.  
Enter an empty property name when done.  
? Property name: reviewer_name  
? Property type:  
    > string  
? Required? N  
? Default value [leave blank for none]: <leave blank>
```

6. Next add the `reviewer_email` property:

```
Let's add another review property.  
Enter an empty property name when done.  
? Property name: reviewer_email  
? Property type:  
    > string  
? Required? N  
? Default value [leave blank for none]: <leave blank>
```

7. Next add the `comment` property:

```
Let's add another review property.  
Enter an empty property name when done.  
? Property name: comment  
? Property type:  
    > string  
? Required? N  
? Default value [leave blank for none]: <leave blank>
```

8. Finally add a property for the item `rating` :

```
Let's add another review property.  
Enter an empty property name when done.  
? Property name: rating  
? Property type:  
    > number  
? Required? Y  
? Default value [leave blank for none]: <leave blank>
```

9. To close the wizard, the next time it asks you to add another review property, just press `Enter` or `Return` to exit.

2.10 - Create a Relationship Between the `item` and `review` Data Models

The next step in this lab is to create a relationship between the `item` model and the `review` model. Even though the models reference entities in entirely different databases, API Connect provides a way to create a logical relationship between them. This logical relationship is then exposed as additional operations for the item model.

1. In the terminal session, type the following command:

```
apic loopback:relation
```

2. Enter the details for the relationship as follows:

```
? Select the model to create the relationship from:  
    > item  
? Relation type:  
    > has many  
? Choose a model to create a relationship with:  
    > review  
? Enter the property name for the relation: reviews  
? Optionally enter a custom foreign key: <leave blank>  
? Require a through model? No
```

2.11 - Verify the Relationship

To verify that the relationship has been created, you will open the API Connect Designer and view the operations on the Explore page.

1. In the terminal session, type the following command to launch the API Connect Designer window:

```
apic edit
```

2. Click on the `inventory` link from the APIs tab.

The screenshot shows the IBM API Connect interface. At the top, there's a blue header bar with the text 'IBM API Connect' and 'inventory'. Below the header, there are four tabs: 'Products', 'APIs' (which is underlined in blue), 'Models', and 'Data Sources'. Under the 'APIs' tab, there are two buttons: '+ Add' and 'Search APIs'. Below these buttons, there's a search bar labeled 'Title'. A red box highlights the entry 'inventory 1.0.0 [inventory.yaml]'. The rest of the interface is mostly empty space.

3. Scroll down to the `Paths` section of the API definition.

Notice how three new API paths have been created which allow access to item review data:

The screenshot shows the 'Paths' section of an API definition. It lists several paths:

- /items/{id}/reviews/{fk}
- /items/{id}/reviews
- /items/{id}/reviews/count
- /items
- /items/{id}/exists

A red box highlights the first three paths: /items/{id}/reviews/{fk}, /items/{id}/reviews, and /items/{id}/reviews/count.

4. Click the `x` button on the Firefox tab or window to close the browser.
5. Select the `Terminal Emulator` from the taskbar to open the command line.
6. Even though we closed the browser, the API Designer application itself is still running.

Hold the `control` key and press the `c` key to end the API Designer session:

`control+c`

This will return you to the command line prompt.

Lab 2 - Validation

Using both the CLI and the API Designer, a lot of files have been created

and updated. Before moving on to the next lab, a simple script has been provided for you which will validate your source files against those of a completed lab. These next steps will help ensure that easily made typing errors do not cause problems down the line.

1. From the `Terminal Emulator`, type:

```
validate_lab 2
```

2. The script will execute a series `diff` commands against specific files in your project folder (`~/ThinkIBM/inventory`)

```
student@xubuntu-vm:~/ThinkIBM/inventory$ validate_lab 2
Running validation check on file: common/models/item.json
... Looks good!

Running validation check on file: common/models/review.json
... Looks good!

Running validation check on file: definitions/inventory.yaml
... Looks good!

Running validation check on file: server/datasources.json
... Looks good!
```

3. If your output looks the same as the image above, then everything matches up and you may continue to lab 3. However, if a discrepancy was found with one of the files, we can look a little closer to see if it is something that will cause a problem.
4. In the example below, an issue was found in the `server/datasource.s.json` file.

```
Running validation check on file: server/datasources.json
12c12
<      "host": "mongo.thinkibm",
-----
>      "host": "mongo.think.ibm",
```

5. The first line specifies the line numbers of the two files where the differences occur. In this example that comes out to line 12 of both files.

6. Next, you will notice a block that specifies what the differences were.

Any lines that begin with a < sign belong to your source files.

And, any lines that begin with a > sign belong to files from our correct sources.

7. In the example provided, the host for the MongoDB server was mistyped. This will cause a problem later when your application attempts to connect to the incorrect host address.

8. For this lab, there are four key files that need validation.

- common/models/item.json **and** review.json
 - Key elements in these file are the element names and data types.
- definitions/inventory.yaml
 - Discrepancies in the item.json or review.json files will show up here in the data model definitions.
- server/datasources.json
 - Most elements in this file must be accurate.

9. If you have differences in one or more of your files, the good news is they are easy to fix. Run the following command to merge the corrected changes into your source files:

```
merge_lab 2
```

Lab 2 - Conclusion

Congratulations! In this lab you learned:

- How to create a multi-model LoopBack application
- How to create a Representational State Transfer (REST) API definition using IBM Connect API Designer
- How to create a Representational State Transfer (REST) API definition using IBM Connect Command Line
- How to use the LoopBack MySQL Connector
- How to use the LoopBack MongoDB Connector
- How to test a REST API
- How to create relationships between models

Lab 3 will build on what you have already created to enable processing hooks and publish the APIs to the API Manager.

Proceed to [Lab 3 - Customize and Deploy an Application](#)

Lab 3 - Customize and Deploy the Inventory Application

In this lab, you will add capabilities into the LoopBack application which was created in Lab 2. You will add custom javascript code which will alter the default behavior of the application. Once your edits are complete, you will package the application and publish it to a WebSphere Liberty runtime collective where it will be managed and enforced by the API Connect solution.

Lab 3 - Objective

In the following lab, you will learn:

- About LoopBack remote hooks
- How to create a remote hook
- How to publish a LoopBack application to a Liberty runtime collective

Lab 3 - Case Study Used in this Tutorial

At this point, you have: created a basic application template, added an `item` data model backed by a MySQL datasource, added a `review` data model backed by a MongoDB data source, added a relationship between the `item` and `review` models.

In this tutorial you will extend the `inventory` application by adding a remote hook. Remote hooks allow you to provide pre- and post-processing to an API call, such as adding additional header information to a remote service or calculating a value, which is what you will do in this lab.

Then, you will publish your LoopBack Inventory application to the Liberty Collective. Making it generally available for consumption.

For more information on [Liberty Collectives](#) see here:

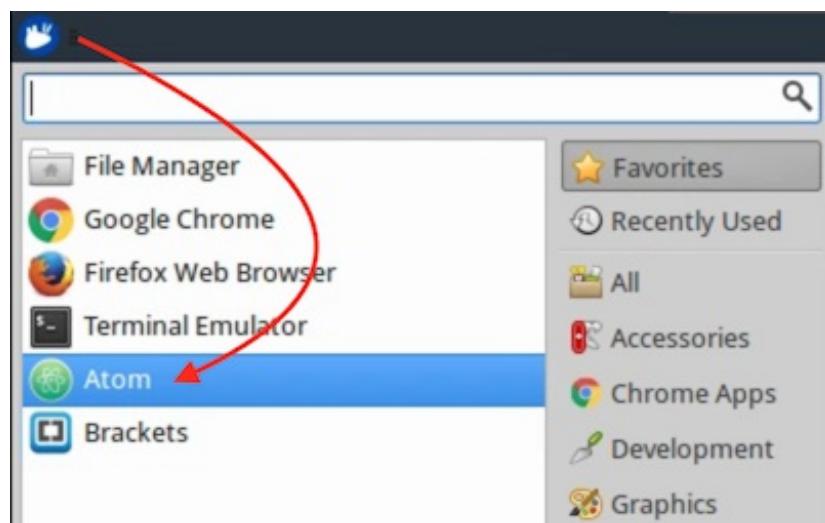
https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/cwlp_collective_arch.html

Lab 3 - Step by Step Lab Instructions

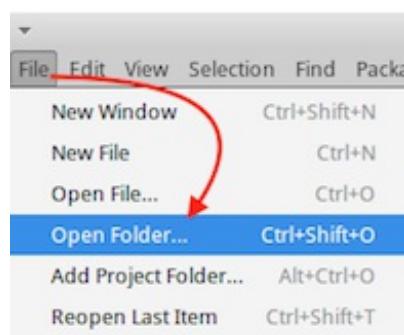
Lab 3.1 - Edit the Application Configuration

Before publishing the API for our application, the configuration file that was generated for you needs to be edited. By default, the generated application uses a base path of `/api`. In the next few steps, you will modify the base path to listen on `/inventory`.

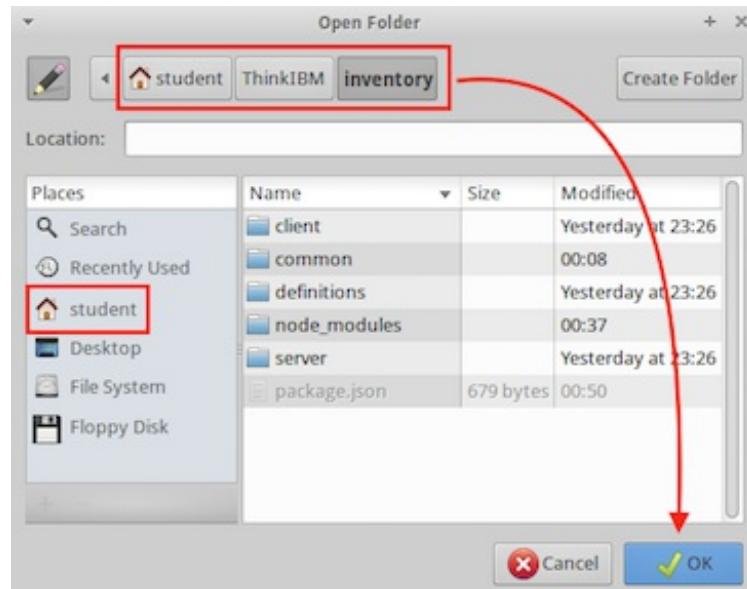
1. Click on the application favorites menu and open the `Atom` text editor.



2. From the `Atom` menu, click on `File > Open Folder`.

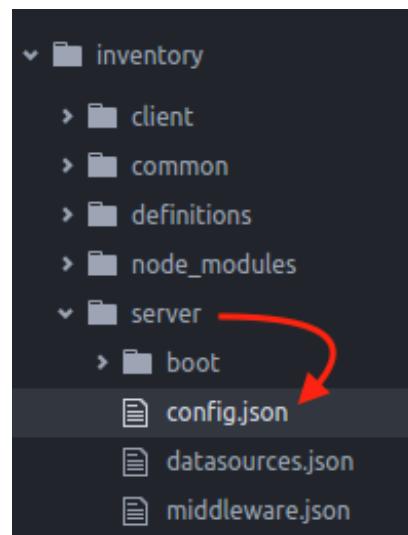


3. Click on the `student` location from the Places menu, then navigate to the `ThinkIBM > inventory` folder and click the `OK` button.



LoopBack applications use a series of configuration files which drive the application runtime. For more information about these files, review the table in Lab 1.

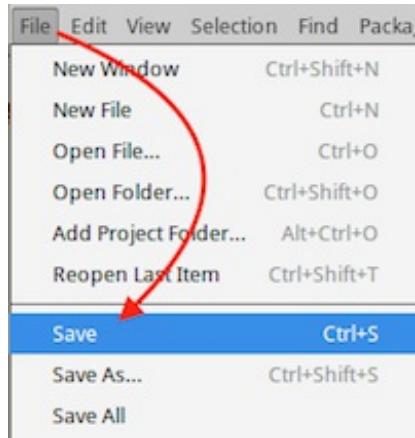
4. From the folder tree menu, expand the `server` folder and click on the `config.json` file to view the source.



5. Edit line 2 of the `config.json` file. Change `/api` to `/inventory`.

```
config.json
1  {
2    "restApiRoot": "/inventory",
3    "host": "0.0.0.0",
4    "port": 3000,
```

6. Use the `Atom` file menu to save the changes.



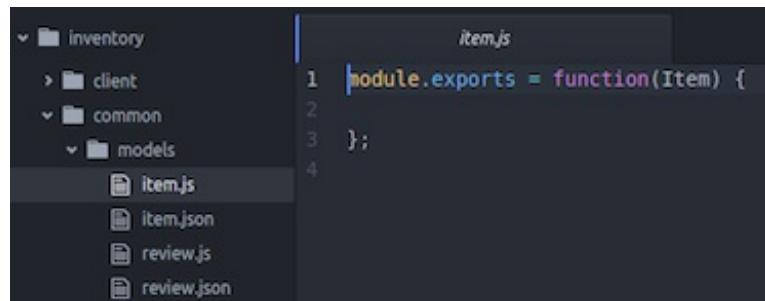
Lab 3.2 - Create a Remote Hook

Remote hooks are custom javascript code that execute before or after calling an operation on a LoopBack application.

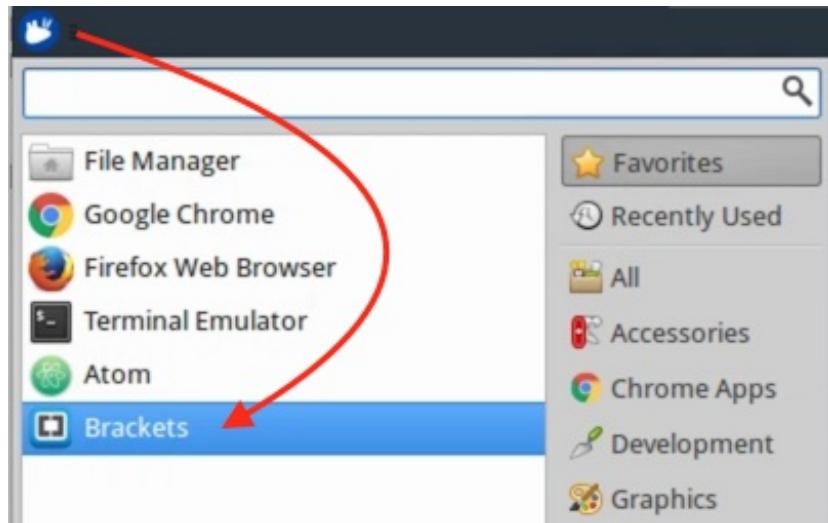
For more information on Remote Hooks please see:

<https://docs.strongloop.com/display/public/LB/Remote+hooks>

1. In the Atom editor, expand the directory structure for the common/models location and select the item.js file.



2. You are going to update this file to include a new remote hook function which will run *after* a new review is submitted for an item. The function will take an average of all reviews for that item, then update the item rating in the MySQL data source.
3. To avoid potential for typing errors, a sample file is available for you to copy. Use the favorites menu to open the Brackets application.



4. Expand the `lab_files/lab3` folder and select the example `item.js` file.

The screenshot shows the Brackets IDE interface. The title bar says "lab3/item.js (lab_files) - Brackets". The menu bar includes File, Edit, Find, View, Navigate, Debug, and Help. Below the menu is a toolbar with icons for file operations. The left sidebar shows a file tree with "Working Files" expanded, showing "item.js" under "lab3". Other folders like "lab_files", "lab2", and "lab4" are also visible. The main editor area contains the following JavaScript code:

```
1 module.exports = function (Item) {
2   Item.afterRemote('prototype.__create__reviews', function (ctx,
3     remoteMethodOutput, next) {
4       // Set up a
5       // function to run after a review is created
6       var itemId = remoteMethodOutput.itemId;
7       // Get the
8       // id of the item that the review was just created for
9       console.log("calculating new rating for item: " + itemId);
```

5. Use the menu option for `Selection > Select All` to highlight all of the text.
6. Use the menu option for `Edit > Copy` to copy the file contents to your clipboard.
7. Return to the `Atom` application. **Remove** everything in the `item.js` file. Then paste (`control+v` or `Edit > Paste`) the contents of your clipboard to update the file.

```
item.js
1 module.exports = function (Item) {
2
3     Item.afterRemote('prototype.__create_reviews', function (ctx, remoteMethodOutput, next) {
4         var itemId = remoteMethodOutput.itemId;
5
6         console.log("calculating new rating for item: " + itemId);
7
8         var searchQuery = {include: {relation: 'reviews'}};
9
10        Item.findById(itemId, searchQuery, function findItemReviewRatings(err, findResult) {
11            var reviewArray = findResult.reviews();
12            var reviewCount = reviewArray.length;
13            var ratingSum = 0;
14
15            for (var i = 0; i < reviewCount; i++) {
16                ratingSum += reviewArray[i].rating;
17            }
18
19            var updatedRating = Math.round((ratingSum / reviewCount) * 100) / 100;
20
21            console.log("new calculated rating: " + updatedRating);
22
23            findResult.updateAttribute("rating", updatedRating, function (err) {
24                if (!err) {
25                    console.log("item rating successfully updated");
26                } else {
27                    console.log("error updating rating for item: " + err);
28                }
29            });
30
31            next();
32        });
33
34    });
35}
```

```

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();
    });
  });
};

```



You can scroll to the right within the Atom text editor to view the code comments describing what each line is doing.

8. Use the File > Save menu option to save the changes.

3.2.1 - Verify the inventory application

Before you publish the API provider application, verify that the inventory application starts correctly.

1. Return to your `Terminal Emulator` session, or open a new one if you had closed it previously.
2. Switch to the `~/ThinkIBM/inventory` directory.

```
cd ~/ThinkIBM/inventory
```

3. Start the Node application with the `npm start` command.

```
npm start
```

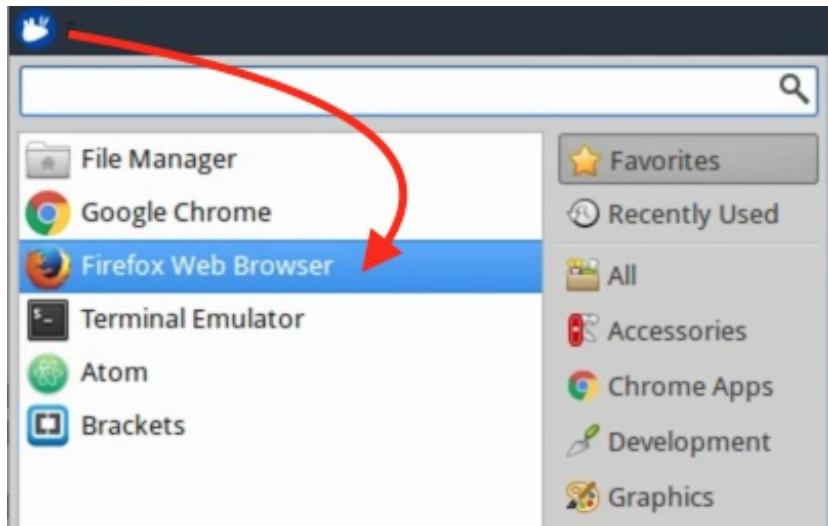
4. Open a web browser to `http://localhost:3000/inventory/items`.
5. Make sure that the API operation call returns a list of items in a JSON object.
6. In the terminal window, press `Ctrl+c` to stop the Node application.

Lab 3.3 - Publish App to a Liberty Collective

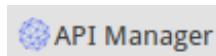
In this section, you will publish the `inventory` application to a Liberty runtime collective for general consumption.

3.3.1 - Register the App with API Connect and Liberty

1. Use the favorites menu to launch the `Firefox Web Browser` :



2. Click on the `API Manager` bookmark:



3. Enter the following credentials and then click the `Sign in` button:

Username: `student@think.ibm`

Password: `Passw0rd!`



Username and Password may already be saved for you by the browser, you can re-use these saved credentials.

IBM API Connect

API Manager

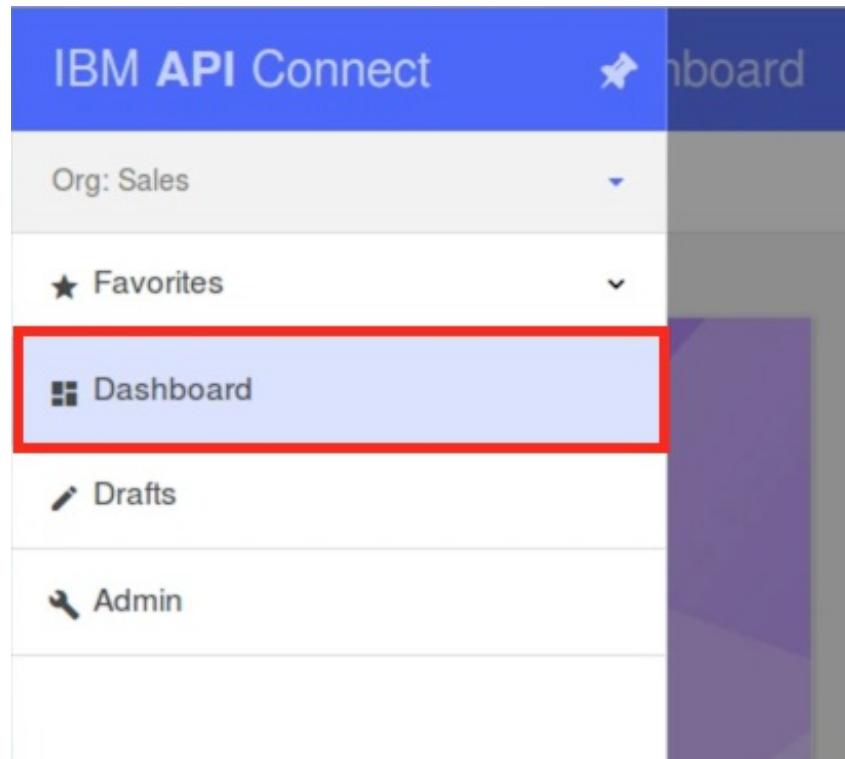
Username
`student@think.ibm`

Password

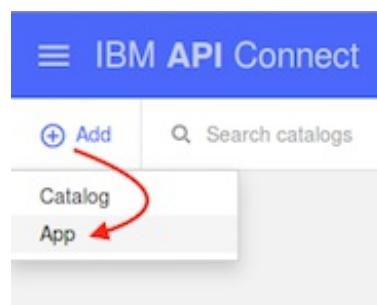
Sign in

Forgot password?

4. Select the menu button from the top left corner of the page.
5. Open the `Dashboard` view.



1. Now that the API Manager Dashboard is open, click the `+ Add` button and select `App` from the list:



2. Fill out the `Add App` form with the following details:

| Display Name: `inventory`

| Name: `inventory`

| Collective: `AppSvr`

Add App

Display Name
Inventory

Name
inventory

Collective
AppSrv

Add **Cancel**

3. Click the **Add** button to link the application between our API Connect server and the Liberty Collective server. This step creates a registration that allows app management from API Connect once the application is published.

3.3.2 - Configure the Developer Toolkit to Communicate with API Connect

1. Click on the **hyperlink** icon inside of the **Inventory** app tile.

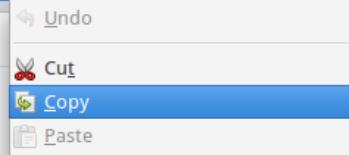


2. Copy the contents of the popup to your system clipboard.

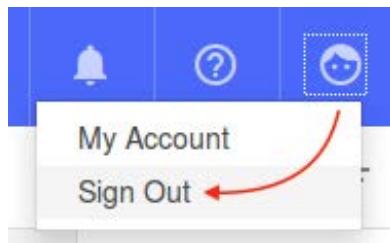
App Identifier

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

```
apic config:set app=apic-app://mgr.think.ibm/orgs/sales/apps/inventory
```



3. Click on the user profile icon and select `Log Out`.



4. Close the Firefox browser by clicking the `x` on the tab or browser window.
5. Return to your `Terminal Emulator` session, or open a new one if you had closed it previously.
6. Ensure you are in the `~/ThinkIBM/inventory` project folder by typing the following command:

```
cd ~/ThinkIBM/inventory
```

7. Use the terminal menu bar to select `Edit > Paste` to paste the contents of your clipboard. If you did not copy the command earlier, you can type it here:

```
apic config:set app=apic-app://mgr.think.ibm/orgs/sales/apps/inventory
```

8. Continue setting up the development environment by logging into the API Connect management server:

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

- a. Use the following credentials when prompted:

5. USE THE FOLLOWING CREDENTIALS WHEN PROMPTED.

Server: mgr.think.ibm

Username: student@think.ibm

Password: Passw0rd!

3.3.3 - Publish the Application

1. Ensure the Liberty Collective server is up and running by typing:

```
wlpn-controller start
```

2. Type the following command to package the `inventory` application and publish it to the collective server:

```
apic apps:publish
```



We used the command line in the previous few steps to publish the application. However, we could have also used the API Designer web experience to accomplish the same task.

3. The terminal will prompt you when the publication is complete:

```
student@xubuntu-vm:~/ThinkIBM/inventory$ apic apps:publish
...preparing project
...building package for deploy
...uploading packages to appsvr.think.ibm:9443, scale: 1
upload successful: inventory-5745df51e4b046e5b9ec4e01-1464197095315-package.tgz
upload successful: inventory-5745df51e4b046e5b9ec4e01-1464197095315.deploy.xml
upload successful: apic.inventory-5745df51e4b046e5b9ec4e01-1464197095315.scalingPolicy.xml

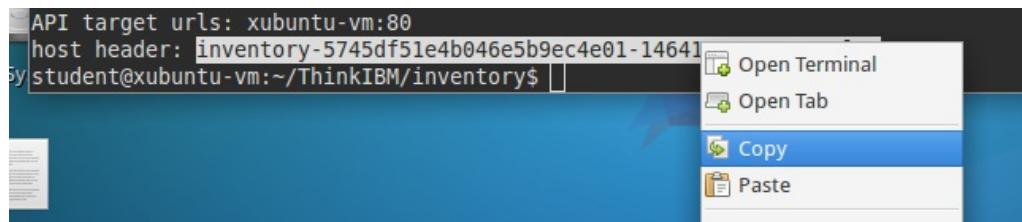
upload to liberty server completed successfully.
Applications may take a few minutes to update and start.

Collectives admin center: https://appsvr.think.ibm:9443/adminCenter.

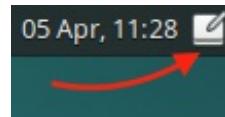
Please update your swagger with:
API target urls: xubuntu-vm:80
host header: inventory-5745df51e4b046e5b9ec4e01-1464197095315.sales
student@xubuntu-vm:~/ThinkIBM/inventory$
```

4. You will need the `host` header that is returned in the next lab.

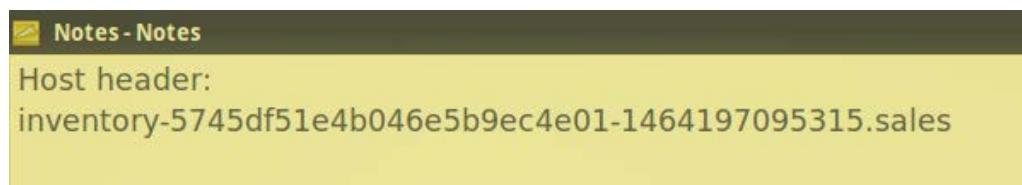
Highlight the `host` header: value and then **right-mouse-click** to show the menu and select `Copy`.



5. Open the `Notes` application by clicking on the notepad icon in the task bar:



6. Paste (use the `control+v` keyboard command) the host header into the `Notes` window. Add a label so that you know what the value is:



Lab 3 - Validation

Using both the CLI and the API Designer, a lot of files have been created and updated. Before moving on to the next lab, a simple script has been provided for you which will validate your source files against those of a completed lab. These next steps will help ensure that easily made typing errors do not cause problems down the line.

1. From the `Terminal Emulator`, type:

```
validate_lab 3
```

2. The script will execute a series `diff` commands against specific files in your project folder (`~/ThinkIBM/inventory`)
3. If the output of the `validate_lab` script includes discrepancies, you may merge the corrected changes into your source files by typing:

```
merge_lab 3
```

Lab 3 - Conclusion

In this lab you learned:

- How to create a remote hook
- How to test the remote hook
- Publish App to Liberty Collective

Proceed to [Lab 4 - Configure and Secure an API](#)

Lab 4 - Configure and Secure an API

In this lab, you will learn how to configure and secure the `inventory` API created during loopback application generation. Using the graphical design tools in API Designer, you will create an OAuth 2.0 provider API called `oauth` and then update the `inventory` API to use this provider. You will use the API Editor assembly view to specify the API's runtime behavior.

Lab 4 - Objective

In the following lab, you will learn:

- How to create an OAuth 2.0 Provider, specifically using the Resource Owner Password grant type.
- How to secure an existing API using the newly created OAuth 2.0 Provider.
- How to add catalog-specific properties to an API.
- How to assemble an API implementation using the activity-log, set-variable and invoke policies

Lab 4 - Case Study Used in this Tutorial

In this tutorial, you will secure the Inventory API to protect the resources exposed by **ThinkIBM**. Consumers of your API will be required to obtain & provide a valid OAuth token before they can invoke the Inventory API.

Lab 4 - Step by Step Lab Instructions

4.1 - Working with the Inventory API in API Designer

1. First, launch API Designer by typing the following commands from your project directory:

```
cd ~/ThinkIBM/inventory/  
apic edit
```

API Designer will open in your browser. You may see an informational message about Draft APIs. (This message appears the very first time you launch the API Designer.) If so, click the `Got it!` button, when you are ready to proceed to creating an API.

You should see the APIs view, and a single API listed. The `inventory` API was automatically created during loopback app generation. We will edit this API at a later step.

The screenshot shows the IBM API Connect interface. At the top, there's a blue header bar with the text "IBM API Connect" and "inventory". On the right side of the header are icons for "Publish", a question mark, and a smiley face. Below the header is a navigation bar with tabs: "Products", "APIs" (which is highlighted in blue), "Models", "Data Sources", "Run", and "Explore". Underneath the navigation bar is a search bar with the placeholder "Search APIs" and a "+ Add" button. The main content area displays a table with three columns: "Title", "Last Modified", and "Type". There is one row in the table representing the "inventory" API. The "Title" column shows "inventory 1.0.0 [inventory.yaml]", the "Last Modified" column shows "3 minutes ago", and the "Type" column has a small trash can icon.

alt text

4.2 - Adding a New OAuth 2.0 Provider API

1. Click the `+ Add` button and select `OAuth 2.0 Provider API` from the menu.
2. Specify the following properties and click the `Next` button to continue.

| Title: `oauth`

| Name: `oauth`

| Base Path: `/oauth20`

| Description: `API for Obtaining Access Tokens`

Provide a title and base path for the new OAuth 2.0 provider.

| | |
|-------------|---------------------------------------|
| Title | oauth |
| Name | oauth |
| Base Path | /oauth20 |
| Version | 1.0.0 |
| Description | API for Obtaining OAuth Access Tokens |

[Cancel](#) [Next](#)

3. Accept the default radio button selection labeled

Don't add to a product and click the Add button.

The API Editor will launch. If this is your first time using the API Editor, you will see an informational message. When you are ready to proceed, click the Got it! button to dismiss the message.

The API Editor opens to the newly created oauth API. The left hand side of the view provides shortcuts to various elements within the API definition: Info, Host, Base Path, etc. By default, the API Editor opens to the Design view, which provides a user-friendly way to view and edit your APIs. You may notice additional tabs labeled Source and Assemble. We will work with these views as well.

The screenshot shows the IBM API Connect interface. At the top, there's a blue header bar with the text "IBM API Connect" and "inventory 1.0.0". Below the header, there's a navigation bar with four tabs: "All APIs", "Design" (which is underlined in blue), "Source", and "Assemble". On the left side, there's a sidebar with several menu items: "Info", "Host", "Base Path", "OAuth 2", "Schemes", "Consumes", "Produces", "Lifecycle", "Policy Assembly", and "Security Definitions". The main content area is titled "Info" and displays the following details for the "oauth" API:

| | |
|-------------|-------|
| Title | oauth |
| Name | oauth |
| Version | 1.0.0 |
| Description | |

4. Navigate to the `Host` section of the API. Remove `$(catalog.host)` from the Host field, as we want to keep this blank.

5. Navigate to the `OAuth 2` section.

Over the next several steps, we will set up OAuth-specific options, such as client type (public vs confidential), valid access token scopes, supported authorization grant types, etc. The [OAuth 2.0 Specification](#) has detailed descriptions of each of the properties we are configuring here.

6. For the Client type field, click the drop down twisty and select `Confidential`.
7. Three scopes were generated for you when the OAuth API Provider was generated: `scope1`, `scope2`, `scope3`.
8. Modify the values for `scope1`, set the following fields:

| Name: `inventory`

| Description: `Access to Inventory API`

9. Delete `scope2` and `scope3` by clicking the trashcan icons to the right of the scope definitions.
10. We want to configure this provider to *only* support the Resource Owner Password Credentials grant type. Deselect the `Implicit`, `Application` and `Access Code Grants`, but leave `Password` checked.

Grants Implicit Password Application Access Code

11. Set the remaining OAuth 2 settings as follows:

| Collect credentials using: `basic`

| Authenticate application users using: `Authentication URL`

| Authentication URL: `https://services.think.ibm:1443/auth`

| TLS Profile: **remove** `tls-profile-4` and leave blank

Deselect the `Enable revocation URL` option

When complete, your settings should look like this:

The screenshot shows the configuration interface for an OAuth client. Key fields highlighted with red boxes include:

- Client type:** Confidential
- Scopes:** inventory (Scope Name), Access to the Inventory API (Description)
- Grants:** Password (selected checkbox)
- Identity extraction:** Basic
- Authentication:** Authentication URL: https://services.think.ibm:1443/auth
- Authorization:** Authenticated
- Tokens:** Enable refresh tokens
- Count:** 2048
- Enable revocation URL:** (checkbox is unchecked)



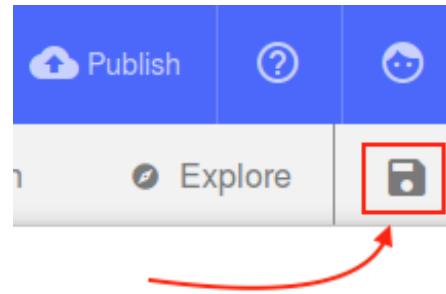
The scope defined here must be identical to the scope that we define later when telling the `inventory` API to use this OAuth config. A common mistake is around case sensitivity. To avoid running into an error later, make sure that your scope is set to all **lowercase**.

12. Navigate to the `Paths` section. Notice that the generated paths begin with `/oauth2`. However, since we have configured our base path to be `/oauth20`, we will shorten the authorization and token paths.
13. Change the `/oauth2/authorize` path to `/authorize`
14. Change the `/oauth2/token` path to `/token`

The screenshot shows the API Editor interface with two defined paths:

- /authorize**: This path has a red box around its path definition. It includes sections for "Parameters" (No parameters defined) and "Operations". Under "Operations", there is a blue "GET" button labeled "/authorize" and a green "POST" button labeled "/authorize", each with a trashcan icon to its right.
- /token**: This path also has a red box around its path definition. It includes sections for "Parameters" (No parameters defined) and "Operations". Under "Operations", there is a green "POST" button labeled "/token" with a trashcan icon to its right.

15. Click the `Save` icon in the top right corner of the editor to save your changes.



4.3 - Configuring and Securing the Inventory API

1. Click the `All APIs` link at the top left of the API Editor to return to list of APIs.
2. Click the `inventory` link.

The inventory API will open in the API Editor, where we can make the necessary configuration changes. Over the next several steps you will set this API up to use the OAuth provider you just created.

3. Click on the `trashcan` icon for the `x-any` Definition to remove it. Confirm the removal by clicking the `OK` button in the prompt.
4. Navigate to the `Base Path` section.

Change the base path from `/api` to `/inventory`.

5. Navigate to the `Host` section of the API. Remove the `$(catalog.host)` value.

As with the OAuth API Provider we just created, we want this value to remain empty.

6. Navigate to the `Security Definitions` section.

Click the `+` icon in the **Security Definitions** section and select `OAuth` from the menu.



A new security definition is created for you, called `oauth-1 (OAuth)`.

7. Scroll down slightly to edit the newly created security definition.

Set it to have the following properties:

Name: `oauth`

Description: `Resource Owner Password Grant Type`

Flow: `Password`

Token URL:

`https://api.think.ibm/sales/sb/oauth20/token`

8. Click the `+` icon in the **Scopes** section to create a new scope. Set the following properties:

Scope Name: `inventory`

Description: `Access to all inventory resources`

9. Navigate to the `Security` section and check the `oauth (OAuth)` checkbox.

Now that the API is secured using our OAuth provider, we can define how the API should behave when called. In the next two sections, we will configure the `inventory` API to call our inventory application which was published at the end of Lab 3.

4.4 - Adding API Properties

Before we add and configure the `inventory` API processing policies, we need to set up a couple of API properties. These are used to hold parameters needed by the API, whose values vary based on which Catalog the API is running in.

For example, we will configure the `inventory` API to invoke a backend service. We want to be able to call different instances of that service based on which Catalog the API is deployed to.

1. Navigate to the `Properties` section and click the `+ Add` icon to create a new property. Give it the following attributes:

Name: `app-server`

Description: `Catalog-specific app server URL`

Default Value: `http://localhost/`

2. Add additional catalog-specific values by clicking the `Add value` link. Specify the following catalog name and value:

Catalog: Sandbox

Value: http://appsvr.think.ibm

The screenshot shows a configuration interface for an 'app-server' property. The 'Property Name' is 'app-server' and the 'Description' is 'Catalog-specific app server host'. Under the 'Catalog' dropdown, 'Sandbox' is selected, and the 'Value' is 'http://localhost/'. A red box highlights the 'app-server' property name and its description. Another red box highlights the 'Sandbox' catalog entry in the dropdown menu.



The Catalog and Value fields are strict. They are case-sensitive and must be typed exactly as shown. You will note that the catalog drop down list will have a default entry there for `sb`. It is important that you type in the `Sandbox` name in the Catalog as indicated above, otherwise you will encounter issues.

3. Create another property by clicking on the `+` icon, named

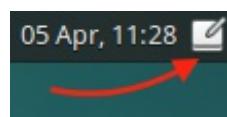
`app-id` :

Name: `app-id`

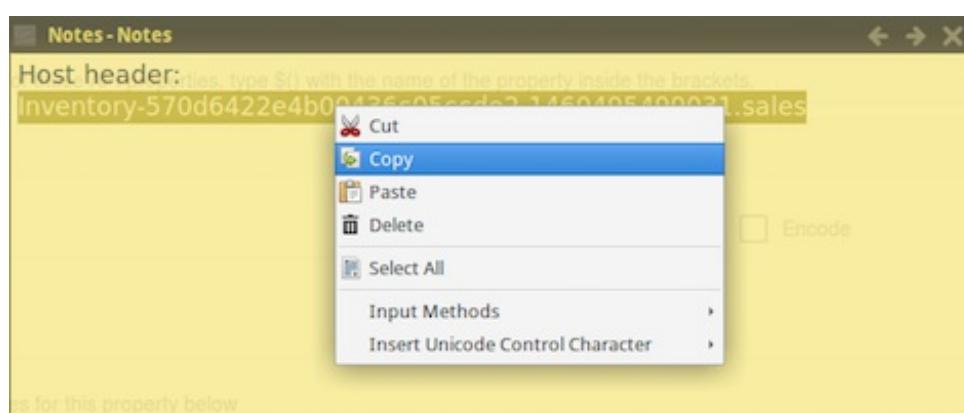
Description: `Inventory application id`

Default Value: `null`

4. Click on the Notepad icon in the top-righthand corner of the OS task bar.



5. Copy the `Host` header from the notepad.



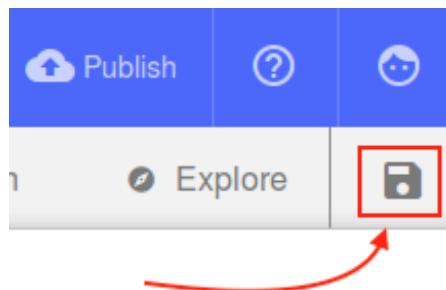
6. Add a new value to the `app-id` property:

Catalog: Sandbox

Value: paste the value of the `app-id` here

The UI may inadvertently add another value after you paste in the app-id. If this occurs, click the trashcan icon to remove the unwanted entry.

7. Save your changes.

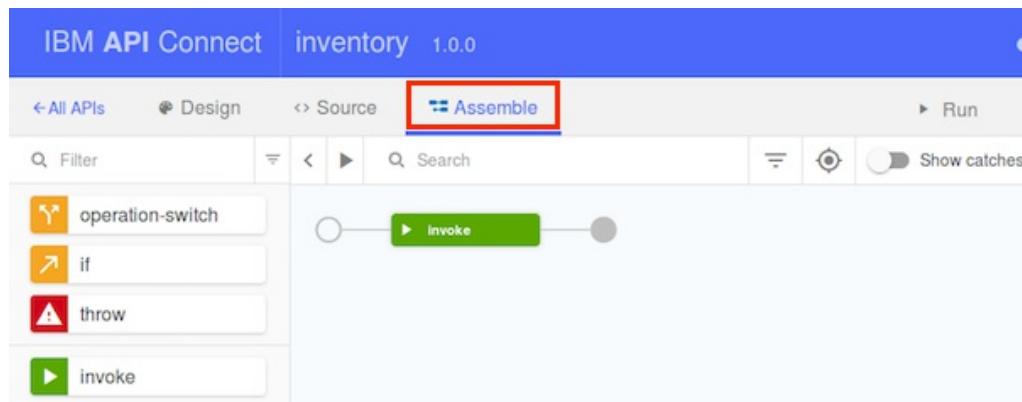


4.5 - Defining API Processing Behavior

An API Assembly provides collection of policies which are enforced and executed on the API Gateway. Policies include actions like modifying the logging behavior and altering the message content or headers.

Additionally, if the out of the box policies do not meet your specific needs, you may opt to create your own policy and have it available for API designers through the API Connect UI.

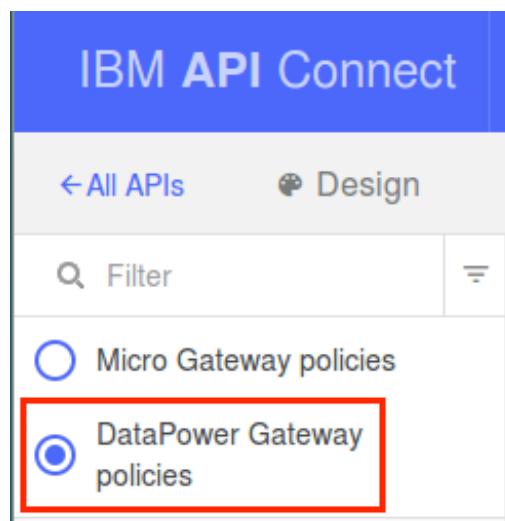
1. Switch to the `Assemble` tab. A simple assembly has been created for you.



2. Modify your assembly to use DataPower Gateway policies.

Expand the `Filter` menu by clicking the menu icon (to the right of the `Filter` label).

Select the `DataPower Gateway policies` radio button.

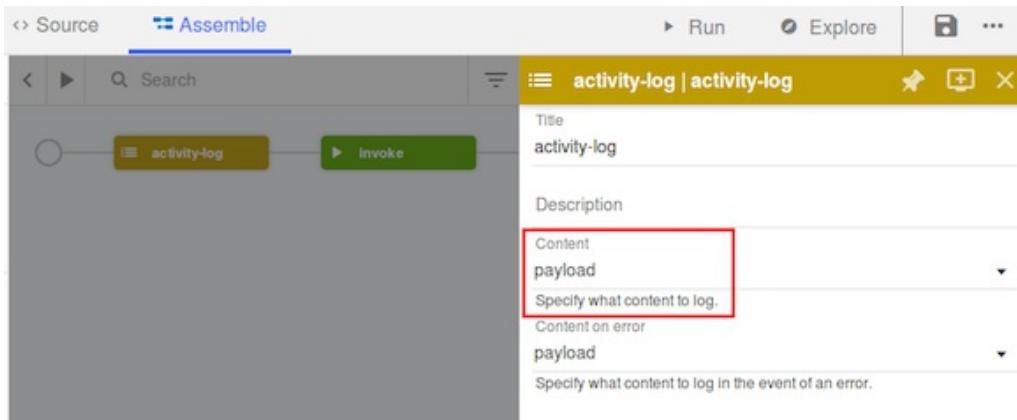


3. Add an `activity-log` policy to the assembly and configure it to log API payload.

Drag the `activity-log` policy from the list of available policies to the left of the `invoke` policy already created in your assembly.

4. Select the newly added `activity-log` step. A properties menu will open on the right of your screen.

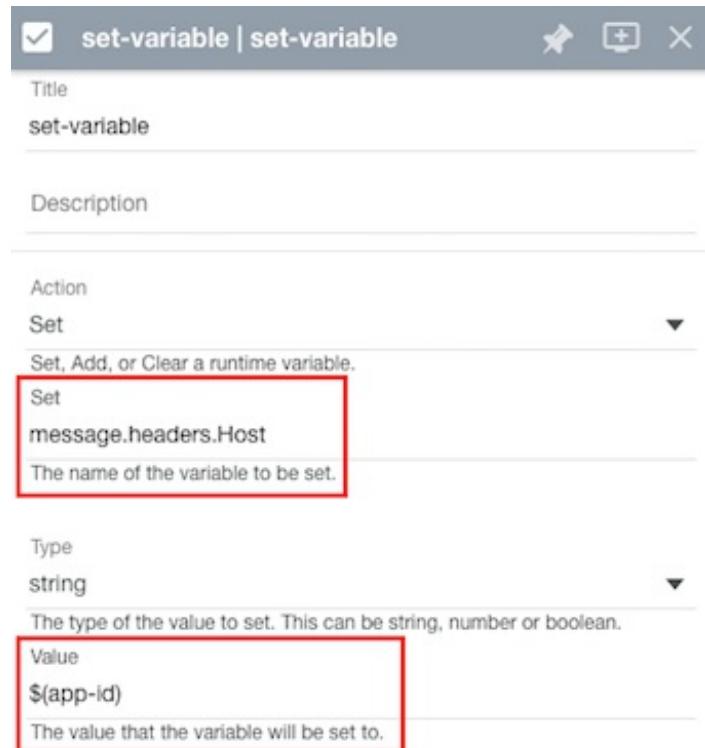
Under `Content` select `payload` from the drop-down list.



5. Click on the `X` to close the activity-log editor menu.
6. Add a `set-variable` policy to the assembly, to the right of the `activity-log` step.
7. Click on the `set-variable` policy to access the editor and set the `Host` header for our request. The `Host` header is used by the liberty runtime to properly route the request to our Inventory application.
8. Click the `+ Action` button.

Edit the action properties to set the header value:

- Action: `Set`
- Set: `message.headers.Host`
- Type: `string`
- Value: `$(app-id)`



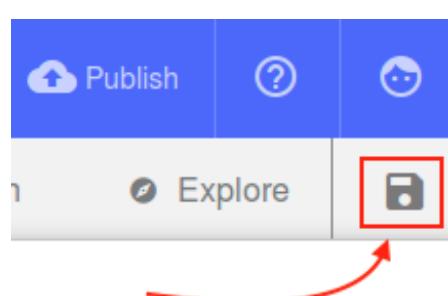
9. Click on the `X` to close the set-variable editor menu.
10. Click on the `invoke` policy to bring up the editor.
11. Replace the default `Invoke URL` path.

Update the URL to be: `$(app-server)$(request.path)`



The `$(request.path)` variable will automatically remove the organization and catalog components of the full API Connect request path. Additionally, the leading slash `/` is already included in the variable value.

12. Click on the `X` to close the invoke policy editor menu.
13. Save your changes.



14. Close the Firefox browser by clicking the `x` on the tab or browser window.
15. Return to your `Terminal Emulator` session.
16. Even though we closed the browser, the API Designer application itself is still running.

Hold the `control` key and press the `c` key to end the API Designer session:

```
control+c
```

This will return you to the command line prompt.

Lab 4 - Validation

Using both the CLI and the API Designer, a lot of files have been created and updated. Before moving on to the next lab, a simple script has been provided for you which will validate your source files against those of a completed lab. These next steps will help ensure that easily made typing errors do not cause problems down the line.

1. From the `Terminal Emulator`, type:

```
validate_lab 4
```

2. The script will execute a series `diff` commands against specific files in your project folder (`~/ThinkIBM/inventory`)
3. If the output of the `validate_lab` script includes discrepancies, you may merge the corrected changes into your source files by typing:

```
merge_lab 4
```

Lab 4 - Conclusion

Congratulations! You have successfully configured and secured the

inventory API. You will consume this API in a later step.

Proceed to [Lab 5 - Advanced API Assembly](#)

Lab 5 - Advanced API Assembly

In this lab, you will learn how to create advanced API assemblies. Using the graphical design tools and source editor, you will create a new API called `financing` which will expose an existing SOAP service as a RESTful API. Additionally, you will create another API called `logistics` which connects to existing public services and uses the assembly tools to map responses into a desired format.

Lab 5 - Objective

In the following lab, you will learn:

- How to create a new API, including object definitions and paths
- How to configure an API to access an existing SOAP service
- How the source editor can be used to import an existing API definition
- How to map data retrieved from multiple API calls into an aggregate response
- How to use gatewayscript directly within an API assembly

Lab 5 - Case Study Used in this Tutorial

In this tutorial, you will expand the product offerings for **ThinkIBM**. In addition to the Inventory API, **ThinkIBM** wishes to provide API's that offer financing and shipping logistics to consumer applications. Your goal is to utilize existing enterprise and public assets to create these API offerings.

Lab 5 - Step by Step Lab Instructions

5.1 - Create the Financing API (REST to SOAP)

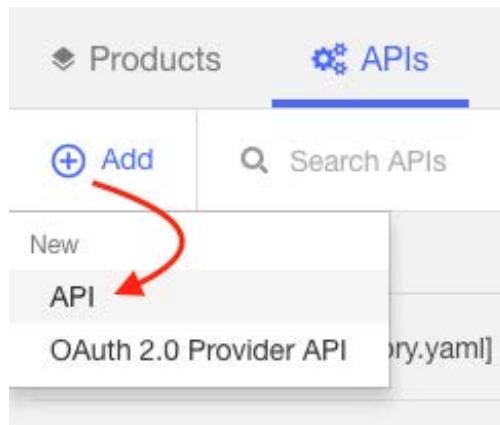
1. If the API Designer screen has not already been launched, open a terminal and start the designer by issuing the following commands:

```
cd ~/ThinkIBM/inventory  
apic edit
```

2. Otherwise, click the `All APIs` link to return to the main Designer screen.

5.1.1 - Create the API Definition

1. Click on the `+ Add` button and select `API`.



2. Fill in the form values for the API, then click the `Next` button to continue.

Title: financing

Name: financing

Base Path: /financing

Version: 1.0.0

Description:

Operations for calculating financing payments

Please provide a title and base path for the new API

Title

financing

Name

financing

Base Path

/financing

Version

1.0.0

Description

Operations for calculating financing payments

Cancel

Next

3. Keep the selected default to `Don't add to a product` and click the `Add` button.
4. API Connect will generate a new swagger definition file for the `financing` API and automatically load the API editor screen. Notice that the API does not contain any paths or data definitions. We will be adding these in the following steps.
5. Click on `Host` from the API editor menu. Remove `$(catalog.host)` from the Host field. We will keep this blank.



The host field will show a red line indicating that the field is required. You may ignore this message.

Host

Host

6. Scroll down to the `Schemes` section, or select it from the API editor menu. Enable the `https` scheme.

Schemes

http

https

wss

ws

7. Below the Schemes section are the `Consumes` and `Produces`

sections. For each of these, choose `application/json`.

The screenshot shows two sections: 'Consumes' and 'Produces'. Both sections have a checked checkbox for 'application/json' and an unchecked checkbox for 'application/xml'. A red box highlights the 'application/json' checkbox in both sections. Below each section is a link to 'Add Media Type'.

8. Next, we need to create the model definition for our new API. These definitions are used in a few places. Their primary role is to serve as documentation in the developer portal on expected input and output parameters; however, they can also be used for data mapping actions. Click on `Definitions` from the API Designer menu.
9. Click the `+` icon in the **Definitions** section to create a new definition. Then, click on `new-definitions-1` to edit the new definition.
10. Edit the `Name` of the definition, set it to `paymentAmount`.
11. The new definition already adds in a sample property called `new-property-1`. Edit the property values:

Property Name: `paymentAmount`

Description: `Monthly payment amount`

Type: `float`

Example: `199.99`

The screenshot shows the 'Definitions' editor for the 'paymentAmount' definition. It displays a table with one row for the 'paymentAmount' property. The 'Name' column is set to `paymentAmount`, which is highlighted with a red box. The 'Type' column is set to `object`. The 'Properties' section below shows a single row for the 'paymentAmount' property, with its value set to `Monthly payment amount`, type set to `float`, and example value set to `199.99`. A red box highlights this row. At the bottom, there is a checkbox for 'Allow additional properties'.

12. Now that we have a definition, we'll create a path. Click on `Paths`

from the API Designer menu.

13. Click on the `+` button to create a new path. The template will generate the path and a GET resource under the path. This is sufficient for our needs, but we could also add other resources and REST verbs to our path if needed.
14. Edit the default path location to be `/calculate`.
 - Recall that our Base Path for this API is `/financing`. This new path will be appended to the base, creating a final path of `/financing/calculate`.
15. Click on the `GET` operation to expand the configuration options for the resource.
16. Each operation needs an Operation ID. Set the Operation ID to: `get.financingAmount`

Operation ID
`get.financingAmount`

17. Next, we have the option of adding request parameters to the operation. This defines the input to the API request. Since this is a GET request, we'll add the required request parameters to the query component of the URI.

Click on the `Add Parameter` link to create a new query parameter. Then, select `Add new parameter` from the sub-menu.

18. We're actually going to need three total parameters for this operation, so go ahead and click on the `Add Parameter` link two more times to add the parameter templates.

| Parameters | | | | | Add Parameter | |
|-----------------|------------|-------------|--------------------------|--------|----------------------------------|---------------------------------------|
| Name | Located In | Description | Required | Type | | |
| api_parameter-1 | Query | | <input type="checkbox"/> | string | <input type="button" value="▼"/> | <input type="button" value="Delete"/> |
| api_parameter-2 | Query | | <input type="checkbox"/> | string | <input type="button" value="▼"/> | <input type="button" value="Delete"/> |
| api_parameter-3 | Query | | <input type="checkbox"/> | string | <input type="button" value="▼"/> | <input type="button" value="Delete"/> |

19. Edit the parameters to set the values:

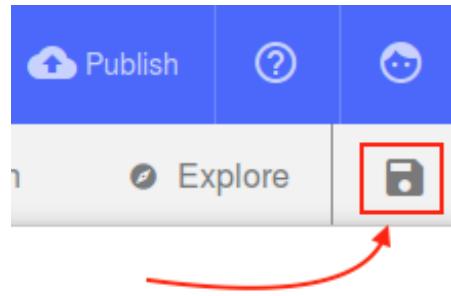
| |
|---------------------------------------|
| Name: amount |
| Located In: Query |
| Description: amount to finance |
| Required: selected |
| Type: float |
| |
| Name: duration |
| Located In: Query |
| Description: length of term in months |
| Required: selected |
| Type: integer |
| |
| Name: rate |
| Located In: Query |
| Description: interest rate |
| Required: selected |
| Type: float |

| Parameters | | | | | Add Parameter |
|------------|------------|--------------------------|-------------------------------------|---------|---------------------------------------------------------------------------------------|
| Name | Located In | Description | Required | Type | |
| amount | Query | amount to finance | <input checked="" type="checkbox"/> | float |  |
| duration | Query | length of term in months | <input checked="" type="checkbox"/> | integer |  |
| rate | Query | interest rate | <input checked="" type="checkbox"/> | float |  |

20. Next we'll set the schema for the response. Since we already defined the `paymentAmount` definition, we will select it from the drop down list. You will find the `paymentAmount` definition at the top of the list.

| Responses | | | Add Response |
|-------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------|--------------|
| Status Code | Description | Schema | |
| 200 | 200 OK | <input type="text" value="paymentAmount"/>  | |

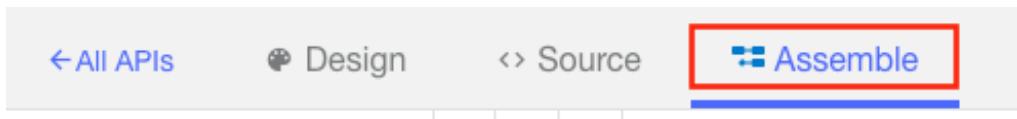
21. Save the API definition.



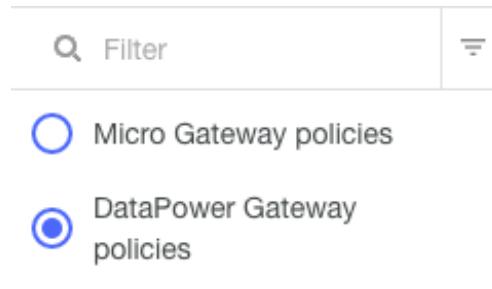
22. You have completed the creation of the new API definition. The path and model data will be presented to our consumers on the developer portal once it's published.

5.1.2 - Build the Financing API Assembly

1. Click on the `Assemble` tab to access the assembly editor.



2. Select the `DataPower Gateway policies` filter.



3. In the policy list, click and drag the `activity-log` policy to the front of the processing pipeline.



4. Click on the `activity-log` action resting on the pipeline to open the configuration options for the policy.

5. Change the selected item for the Content field from `activity` to `payload`.

The screenshot shows the 'activity-log' policy configuration in the WSO2 API Manager. The 'Content' dropdown is open, and 'payload' is selected. A red box highlights this selection. Below, under 'Content on error', 'payload' is also selected.

6. Click the **X** to close the policy editor window to return to the assembly processing pipeline.
7. Now we are going to add the remaining policies required for mapping our REST API into SOAP.
8. Between the `activity-log` and `invoke` policies, add a `map` policy.
9. After the `invoke` policy, add a `gatwayscript` policy and then a `xml-to-json` policy.



10. In order to consume a SOAP-based service from our REST-based API, we need to map the query parameter inputs that we defined as part of the `GET /calculate` operation to a SOAP payload. To do so, click on the `map` policy on our pipeline to open the map editor.
11. Click on the **+** icon to make the editor window fill the screen.



12. On the **Input** column, click on the **pencil** icon to bring up the input editor.



13. Recall that our GET operation has three required query parameters: `amount`, `duration` and `rate`. Click on the `+ input` button three times to add the entries to the input table.
14. Fill in the values for each of the input parameters:

Context variable: `request.parameters.amount`

```

| Name: amount
|
| Content type: none
|
| Definition: float
|
|-----|
|
| Context variable: request.parameters.duration
|
| Name: duration
|
| Content type: none
|
| Definition: integer
|
|-----|
|
| Context variable: request.parameters.rate
|
| Name: rate
|
| Content type: none
|
| Definition: float

```

The screenshot shows the Map Editor interface with a green header bar containing a back arrow, the word "map", and a close button. Below the header, there are two sections: "Title" and "Description".

Title: map

Description:

| | |
|-----------------------------------------------|---------------------|
| Context variable request.parameters.amount | Name amount |
| Content type none | Definition float |

| | |
|-------------------------------------------------|-----------------------|
| Context variable request.parameters.duration | Name duration |
| Content type none | Definition integer |

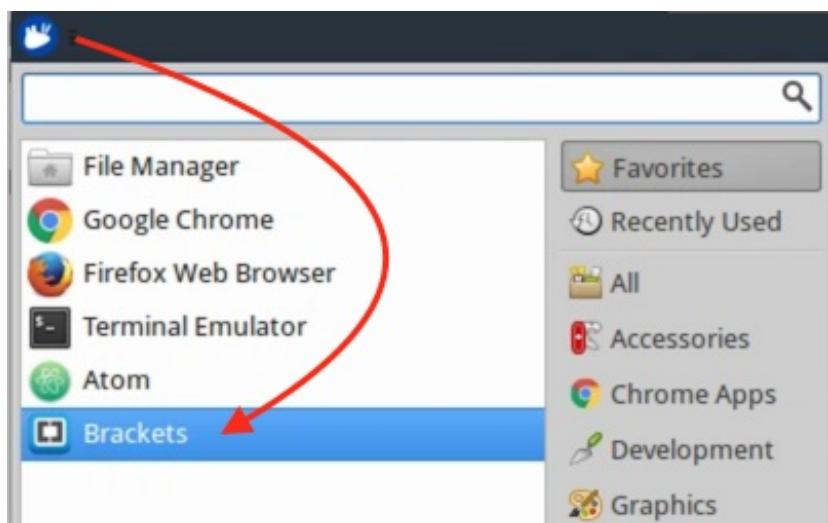
| | |
|---------------------------------------------|---------------------|
| Context variable request.parameters.rate | Name rate |
| Content type none | Definition float |

At the bottom left is a blue "+ input" button, and at the bottom right is a blue "Done" button.

15. Click on the **Done** button to return to the map editor.
16. Next, in the **Output** column, click on the **pencil** icon to add an output definition.
17. Click on **+ output**.

18. Set the `Content type` to `application/xml`.
19. For the `Definition` field, click on the drop down menu and scroll to the bottom to select `Inline schema`.
20. In the interest of time, and to avoid typing errors, a sample schema file has been provided for you.

Open the `Brackets` application from the system task bar, or using the favorites menu.



1. Use the file tree menu to open the `lab5/schema_financingSoap.yaml` file:

 A screenshot of the Brackets code editor. The title bar says "lab5/schema_financingSoap.yaml (lab_files) - Brackets". The file tree on the left shows a directory structure with "lab_files" expanded, showing "lab2", "lab3", "lab4", "lab5" (expanded), "complete", "api_logistics.yaml", "gws_formatMapsLink.js", "schema_financingSoap.yaml" (highlighted with a blue selection bar), and "schema_shippingSvc.yaml". The main editor area shows the YAML content of "schema_financingSoap.yaml".

```

$schema: 'http://json-schema.org/draft-04/schema#'
id: 'http://services.think.ibm'
type: object
properties:
  Envelope:
    type: object
    xml:
      prefix: soapenv
      namespace: 'http://schemas.xmlsoap.org/soap/envelope/'
    properties:
      Body:
        type: object
        properties:
          financingRequest:

```

2. Copy the contents of the `schema_financingSoap.yaml` file to the system clipboard.

```

$schema: 'http://json-schema.org/draft-04/schema#'
id: 'http://services.think.ibm'
type: object
properties:
  Envelope:
    type: object

```

```
type: object
xml:
  prefix: soapenv
  namespace: 'http://schemas.xmlsoap.org/soap/envelope/'
properties:
  Body:
    type: object
    properties:
      financingRequest:
        type: object
        xml:
          prefix: ser
          namespace: 'http://services.think.ibm'
        properties:
          amount:
            id: 'http://services.think.ibm/Envelope/Body/financingRequest/amount'
            type: number
            name: amount
          duration:
            id: 'http://services.think.ibm/envelope/body/financingRequest/duration'
            type: integer
            name: duration
          rate:
            id: 'http://services.think.ibm/envelope/body/financingRequest/rate'
            type: number
            name: rate
        additionalProperties: false
        required:
          - amount
          - duration
          - rate
        name: financingRequest
      additionalProperties: false
      required:
        - financingRequest
    name: Body
    additionalProperties: false
    required:
      - Body
    name: Envelope
  additionalProperties: false
  required:
    - Envelope
title: output
```

3. Switch back to the `Firefox` browser and paste (`control+v`) the SOAP schema definition into the schema editor window, then click the `Done` button.

Provide a schema

Enter YAML or switch to JSON

```
30         type: number
31         name: rate
32         additionalProperties: false
33     required:
34     - amount
35     - duration
36     - rate
37     name: financingRequest
38     additionalProperties: false
39   required:
40   - financingRequest
41   name: Body
42   additionalProperties: false
43   required:
44   - Body
45   name: Envelope
46   additionalProperties: false
47   required:
48   - Envelope
49   title: output
50
```

`Done` `Cancel`

4. Click the `Done` button in the output editor window to return to the map editor.
5. Click on the curly braces `{}` for the `financingRequest: {}` element to expand the view and see the SOAP input elements.

```
        output {
          Envelope:{  
            Body:{  
                financingRequest:{}  
            }  
        }
```

6. Click the '+' icon in menu bar on the the map properties window to maximize the properties, or you might not see the output variables to be mapped.
7. For each of the `Input` query parameters, map them to their respective SOAP `Output` elements.

To map from an input field to an output field, click the circle next to the `source` element once, then click the circle next to the `target` element. A line will be drawn between the two, indicating a mapping from the source to the target.



8. Click the **X** button in the map editor to return to the policy pipeline.
9. Click the **invoke** policy to open its editor.
10. Set the **Invoke URL** to:

`https://services.think.ibm:1443/financing`

The screenshot shows the 'invoke' policy editor. The 'Title' field is empty. The 'Description' field contains the 'Invoke URL' section, which is highlighted with a red box. The URL `https://services.think.ibm:1443/financing` is listed as the invoke URL.

11. Scroll down and set the **HTTP Method** to **POST**.

HTTP Method
POST

The HTTP Method to use for the Invoke. The default value is 'GET'.

12. Click the **X** button to return to the policy pipeline.
13. Click on the **gatewayscript** policy to open the editor. Add the following line:

```
session.name('_apimgmt').setVar('content-type', 'application/xml');
```

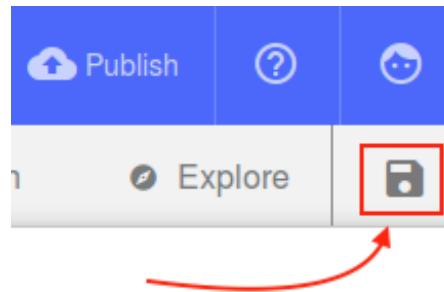
The screenshot shows the 'gatewayscript' policy editor. The 'Title' field is 'gatewayscript'. The 'Description' field is empty. In the script area, the following line of code is present:

```
1 session.name('_apimgmt').setVar('content-type', 'application/xml');
```

14. You are now finished with the assembly for the `financing` API. The assembly takes the following actions:

- Logs requests.
- Maps the REST query parameters into a SOAP body.
- Sets the SOAPAction header.
- Invokes the SOAP service.
- Transforms the SOAP service's response into JSON.

15. Save the API definition.



5.2 - Add Logistics API (Advanced Assembly)

In this lab section, we will be adding a new API called `logistics` which will provide helper services around calculating shipping rates and locating nearby stores.

Rather than require you to build the entire API from scratch again, you will see how you can use the source editor to paste in an API definition that has already been started for you.

5.2.1 - Import Predefined API Definition

1. Click on the `All APIs` link to return to the main API Designer screen.
2. Click on the `+ Add` button to add a new API. Specify the following values:

Title: `logistics`

Name: `logistics`

Base Path: `/logistics`

Version: `1.0.0`

Description: `Provides logistics for sales`

Please provide a title and base path for the new API

Title
logistics

Name
logistics

Base Path
/logistics

Version
1.0.0

Description
Provides logistics for sales

Cancel Next

3. Keep the default `Don't add to a product` option selected and click the `Add` button.
4. Switch to the `Source` tab and look at the API Definition. Any edits you make directly to the source will be immediately visible on the `Design` tab.
5. From the `Brackets` editor, open the `lab5/api_logistics.yaml` file.
Select the full contents of the file and **copy** the contents to the clipboard.
6. Return to the browser, select the contents of the `Source` tab and delete the contents.
7. Paste the contents of your clipboard to update the API Definition.

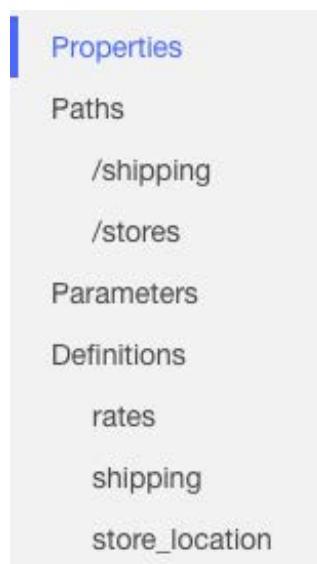
```

1  swagger: '2.0'
2  info:
3    x-ibm-name: logistics
4    title: logistics
5    version: 1.0.0
6  schemes:
7    - https
8  basePath: /logistics
9  consumes:
10   - application/json
11  produces:
12   - application/json
13  securityDefinitions:
14    clientID:
15      description: ''
16      in: header
17      name: X-IBM-Client-Id
18      type: apiKey
19  security:
20    - clientID: []
21  x-ibm-configuration:
22    testable: true
23    enforced: true
24    cors:
25      enabled: true
26    gateway: datapower-gateway
27    catalogs:
28      apic-dev:
29        properties:
30          runtime-url: ${TARGET_URL}
31    properties:
32      shipping_svc_url:
33        value: 'http://shipping.think.ibm:5000/calculate'
34        description: Location of the shipping calculator service
35        encoded: false

```

8. Switch back to the `Design` view and note the changes that have been imported, in particular:

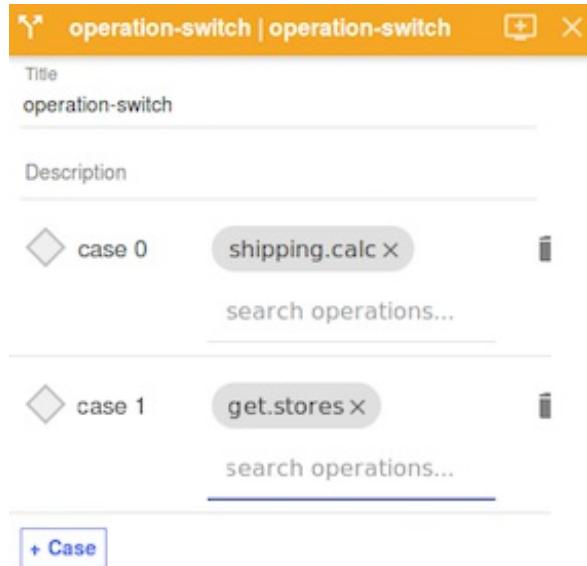
- Properties
- Paths
- Definitions



5.2.2 - Create the Logistics API Assembly

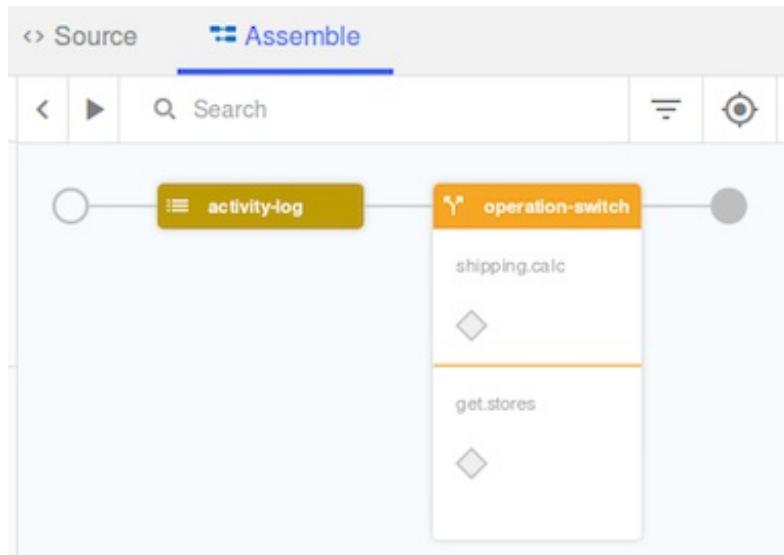
1. Switch to the `Assemble` tab and click the `Create assembly` button.
2. Add an `activity-log` policy to the assembly pipeline.

3. Click the `activity-log` policy to open editor, configure it to log the `payload` for the **Content** field.
4. Add an `operation-switch` policy to the right of the activity-log step.
5. Click on the `operation-switch` policy to launch the editor. A single case `case 0` is created by default.
6. Click `search operations...` to bring up the drop-down list of available operations.
7. Select the `shipping.calc` operation.
8. Click the `+ Case` button to add a second case for the `get.stores` operation.



9. Click the `X` to close the operation switch configuration editor.

You should see two new processing pipelines created on your `operation-switch` step - one for each case:



5.2.2.1 - Configure the `shipping.calc` Case:

This operation will end up invoking two separate back-end services to acquire shipping rates for the respective companies, then utilize a map action to combine the two separate responses back into a single, consolidated message for our consumers.

1. Add an invoke policy to the `shipping.calc` case with the following properties:

Title: `invoke_xyz`

Invoke URL:

```
$(shipping_svc_url)?company=xyz&from_zip=90210&to_zip={zip}
```

Response object variable (scroll to the bottom): `xyz_response`

▶ invoke_xyz | invoke
✖️
➕
✖

Title
`invoke_xyz`

Description
The Invoke URL to be used.

Response object variable
`xyz_response`

The name of a variable that will be used to store the response data from the request. This can then be referenced in other actions, such as 'Map'.

The `{zip}` parameter provided here is a reference to the `zip` parameter defined as input to the operation. The `{zip}` portion of the URL will get replaced by the actual zip code provided by then API consumers.

2. Add a second invoke policy to the `shipping.calc` case with the following properties:

Title: `invoke_cek`

Invoke URL:

```
$(shipping_svc_url)?company=cek&from_zip=90210&to_zip={zip}
```

Response object variable: `cek_response`

The screenshot shows the 'invoke_cek | invoke' configuration dialog. It has a green header bar with a back arrow and a save icon. Below the header, there are two input fields: 'Title' containing 'invoke_cek' and 'Description' containing 'The Invoke URL to be used.' A red box highlights the 'Invoke URL' field, which contains the value '\$(shipping_svc_url)?company=cek&from_zip=90210&to_zip={zip}'. Another red box highlights the 'Response object variable' field, which contains 'cek_response'. A note below the response variable field states: 'The name of a variable that will be used to store the response data from the request. This can then be referenced in other actions, such as 'Map'.'

3. Add a `map` policy after the last invoke, then click it to open the editor.
4. Click the `pencil` icon next to `Input` and specify the following map properties:

Title: `map_responses`

Description:

```
Map responses from invoke_xyz and invoke_cek to output
```

The screenshot shows the 'map_responses' schema configuration window in Brackets. At the top, there's a title bar with the title 'map_responses | map'. Below the title bar, there are fields for 'Title' (set to 'map_responses') and 'Description' (set to 'Map responses from invoke_xyz and invoke_cek to output'). At the bottom right are two buttons: '+ input' (highlighted in blue) and 'Done'.

5. Click the `+ input` button to add an input. Specify the following input configuration:

- Context variable: `xyz_response.body`
- Name: `xyz`
- Content type: `application/json`
- Definition: `Inline schema`

6. After you select `Inline schema`, you will be prompted to "Provide a schema".

Use the `Brackets` text editor to open the `lab5/schema_shippingSvc.yaml` file.

Copy the contents to your clipboard and paste them into the schema editor window.

7. Click `Done` to close the Schema window

The screenshot shows the 'Provide a schema' dialog in Brackets. It has a text area containing YAML code for a schema definition. At the bottom right are 'Done' and 'Cancel' buttons.

```
8   type: object
9   name: company
10  rates:
11    id: 'http://jsonschema.net/rates'
12    type: object
13    properties:
14      next_day:
15        id: 'http://jsonschema.net/rates/next_day'
16        type: string
17        name: next_day
18      two_day:
19        id: 'http://jsonschema.net/rates/two_day'
20        type: string
21        name: two_day
22      ground:
23        id: 'http://jsonschema.net/rates/ground'
24        type: string
25        name: ground
26    name: rates
27  required:
28    - company
29    - rates
```

8. Click the `+input` button again to add another input. Specify the following input configuration:

- Context variable: `cek_response.body`

Name: cek

Content type: application/json

Definition: Inline schema

9. Paste the same schema definition that was used for the previous input (for our lab purposes, the responses from the shipping service are in the same format, thus using the same schema)
10. You now have two inputs assigned to the map policy:

The screenshot shows the 'map' configuration interface. It displays two input fields, each with a context variable, name, and content type. The first input field has a context variable of 'xyz_response.body', a name of 'xyz', and a content type of 'application/json'. The second input field has a context variable of 'cek_response.body', a name of 'cek', and a content type of 'application/json'. Both fields have their 'Definition' set to 'inline'. At the bottom left is a '+ input' button, and at the bottom right is a 'Done' button.

| | |
|---------------------------------------|----------------------|
| Context variable xyz_response.body | Name xyz |
| Content type application/json | Definition inline |

| | |
|---------------------------------------|----------------------|
| Context variable cek_response.body | Name cek |
| Content type application/json | Definition inline |

+ input Done

11. Click the Done button to return to the editor.
12. Click the pencil icon next to Output, then click the + output button to add an output field with the following properties:

Context variable: message.body

Name: output

Content type: application/json

Definition: #/definitions/shipping

The screenshot shows the 'map_responses | map' configuration interface. It displays an output field with a context variable, name, content type, and definition. The output field has a context variable of 'message.body', a name of 'output', a content type of 'application/json', and a definition of '#/definitions/shipping'. At the bottom left is a '+ output' button, and at the bottom right is a 'Done' button.

| | |
|----------------------------------|--------------------------------------|
| Context variable message.body | Name output |
| Content type application/json | Definition #/definitions/shipping |

+ output Done

13. Click the Done button to return to the editor.
14. Complete the mapping. To map from an input field to an output field,

click the circle next to the *source* element once, then click the circle next to the *target* element. A line will be drawn between the two, indicating a mapping from the source to the target.

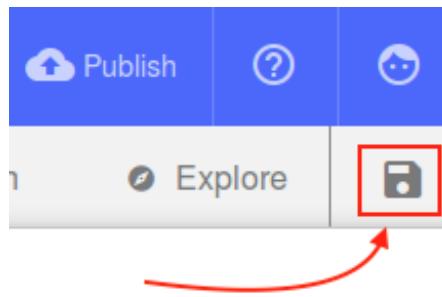


15. Click the to close the map editor.

Your assembly policy for the `shipping.calc` operation is now complete.



16. Save your changes.



5.2.2.2 - Configure the `get.stores` Case:

This operation will call out to the Google Geocode API to obtain location information about the provided zip code, then utilize a simple gatewayscript to modify the response and provide a formatted Google

Maps link.

1. Add an invoke policy to the `get.stores` case with the following properties:

Title: `invoke_google_geolocate`

Invoke URL:

`https://maps.googleapis.com/maps/api/geocode/json?address={zip}`

Response object variable (scroll to the bottom):

`google_geocode_response`

The screenshot shows the configuration for an 'invoke' policy named 'invoke_google_geolocate'. The 'Title' field is set to 'invoke_google_geolocate'. The 'Description' field contains the 'Invoke URL' which is 'https://maps.googleapis.com/maps/api/geocode/json?address={zip}'. The 'Response object variable' is set to 'google_geocode_response'. A note below the response variable states: 'The name of a variable that will be used to store the response data from the request. This can then be referenced in other actions, such as 'Map''. The entire configuration window has a red border around the title and URL fields.

2. Add a `gatewayscript` policy with the following properties:

Title: `gws-format-maps-link`

Paste the contents of `lab5/gws_formatMapsLink.js` into the code section of the policy.

The screenshot shows the configuration for a 'gatewayscript' policy named 'gws-format-maps-link'. The 'Title' field is set to 'gws-format-maps-link'. The 'Description' field contains the following JavaScript code:

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

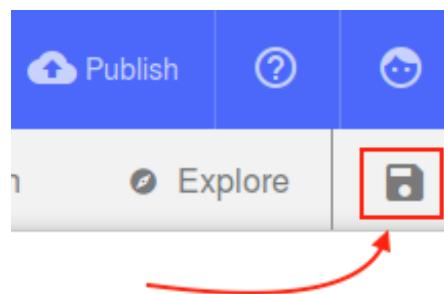


Take a quick look at line 5. Notice how our gateway script file is reading the body portion of the `google_geocode_response` variable which was assigned to the output of the `invoke` action.

3. Click the `X` to close the gatewayscript editor.
4. Your assembly for the `logistics` API will now include two separate operation policies:



5. Save your changes.



6. Close the Firefox browser by clicking the `x` on the tab or browser window.
7. Return to your `Terminal Emulator` session.
8. Even though we closed the browser, the API Designer application itself is still running.

Hold the `control` key and press the `c` key to end the API Designer session:

control+c

This will return you to the command line prompt.

Lab 5 - Validation

Using both the CLI and the API Designer, a lot of files have been created and updated. Before moving on to the next lab, a simple script has been provided for you which will validate your source files against those of a completed lab. These next steps will help ensure that easily made typing errors do not cause problems down the line.

1. From the `Terminal Emulator`, type:

```
validate_lab 5
```

2. The script will execute a series `diff` commands against specific files in your project folder (`~/ThinkIBM/inventory`)
3. If the output of the `validate_lab` script includes discrepancies, you may merge the corrected changes into your source files by typing:

```
merge_lab 5
```

Lab 5 - Conclusion

Congratulations! You have successfully configured two new API's with advanced assemblies. In the next lab, you will bundle the API's into a Product and publish it to the consumer portal.

Proceed to [Lab 6 - Working with API Products](#)

Lab 6 - Working with API Products

API's published by API Connect are bundled into an object called a **Product**. The Product combines one or more API's with one or more Plans.

A **Plan** is effectively a contract between the API Provider and API Consumer which specifies the allowed rate of API calls over a given period of time.

Lab 6 - Objective

In the following lab, you will learn:

- How to create a Product
- How to attach APIs to a Product
- How to create a Plan
- How to publish a Product

Lab 6 - Case Study Used in this Tutorial

Your work as an Application Developer / API Designer is now complete. It's time to switch roles and become an API Product Manager. The role of the API Product Manager is to take the developed assets and bundle them together using a go-to-market strategy.

In the case of **ThinkIBM**, you will publish all of our API's together as a single product offering to API Consumers. Additionally, you will create two plans which have different levels of access to your APIs.

Lab 6 - Step by Step Lab Instructions

6.1 - Creating an API Product

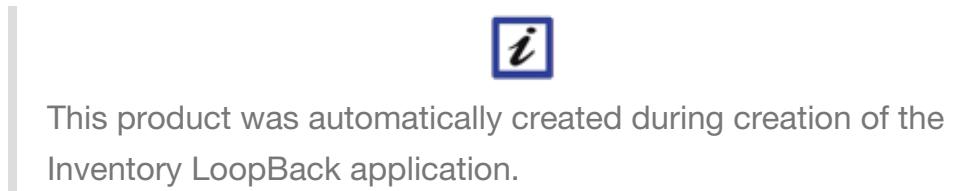
1. If the API Designer screen has not already been launched, open a terminal and start the designer by issuing the following commands:

```
cd ~/ThinkIBM/inventory  
apic edit
```

2. Switch to the Products tab



3. Click the link for the inventory product.



4. Edit the Product Info & Contact details:

- || Title: think
- || Name: think
- || Description:
The **think** product will provide really awesome APIs to your application.
- || Contact Name: Thomas Watson
- || Contact Email: thomas@think.ibm
- || Contact URL: <http://www.ibm.com>

| | |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Info | <p>Title think</p> <hr/> <p>Name think</p> <hr/> <p>Version 1.0.0</p> <hr/> <p>Description The "think" product will provide really awesome APIs to your application.</p> |
| Contact | <p>Name Thomas Watson</p> <hr/> <p>Email thomas@think.ibm</p> <hr/> <p>URL http://www.ibm.com</p> |

5. Specify a License and Terms of Service:

- License Name: The MIT License (MIT)
- License URL: <https://opensource.org/licenses/MIT>
- Terms of Service: paste the contents of the lab6/license.txt file

| | |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| License | <p>Name The MIT License (MIT)</p> <hr/> <p>URL https://opensource.org/licenses/MIT</p> |
| Terms of Service | <p>Terms of Service Copyright (c) 2016+ IBM</p> <p>Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:</p> <p>The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.</p> <p>THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.</p> |

6. Modify the Visibility so that the think product is only visible to Authenticated users :

7. Navigate to the APIs section. Click the **+** button to add all of our new APIs to this product.
8. Check the checkboxes next to `financing`, `logistics` and `oauth`, ensuring that `inventory` is left selected.

| | |
|-----------------------------------------------|-------|
| <input checked="" type="checkbox"/> financing | 1.0.0 |
| <input checked="" type="checkbox"/> inventory | 1.0.0 |
| <input checked="" type="checkbox"/> logistics | 1.0.0 |
| <input checked="" type="checkbox"/> oauth | 1.0.0 |

Cancel **Apply**

9. Click the **Apply** button.
10. Navigate to the Plans section. Modify the default plan by selecting it and specifying the following properties:

- || Title: `Silver`
- || Name: `silver`
- || Description: `Limited access to these APIs`

| Title | Name | Description |
|--------|--------|------------------------------|
| Silver | silver | Limited access to these APIs |

Rate limit (calls / time interval)
 Unlimited
 100 / 1 Hour Enforce hard limit

Approval
 Require subscription approval

11. Click the **+** button to create a new plan. Give it the following properties:

Title: Gold

Name: gold

Description:
Unlimited access to these APIs for approved users

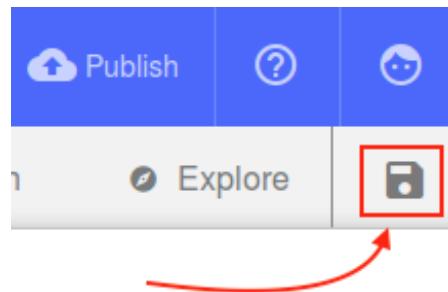
Rate limit: Unlimited

Approval: check Require subscription approval

Plans

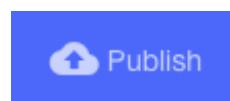
| Title Gold | Description Unlimited access to these APIs for approved users | |
|---------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| Name gold | Rate limit (calls / time interval) <input checked="" type="checkbox"/> Unlimited | Approval <input checked="" type="checkbox"/> Require subscription approval |

12. Save your changes.



6.2 - Publishing the API Product

1. Click the Publish icon.



2. Select Add and Manage Targets from the menu.

3. Select Add a different target.

Publish



4. Provide connection information to sign into the IBM API Connect management server, then click the **Sign in** button:

API Connect host address: `mgr.think.ibm`
Username: `student@think.ibm`
Password: `Passw0rd!`

Sign in to IBM API Connect

API Connect host address
`mgr.think.ibm`

Username
`student@think.ibm`

Password
.....

Back **Cancel** **Sign in**

5. On the **Select an organization and catalog** screen, choose the `Sandbox` catalog and click the **Next** button.

Select an organization and catalog

Organization
`Sales`

Search

None

`Sandbox`

6. On the **Select an App** screen, choose the `Inventory` application and click the **Save** button.

Select an App

A screenshot of a user interface titled "Select an App". At the top is a search bar with the placeholder "Search". Below it is a list of apps. The first item, "Inventory", is highlighted with a red rectangular border around its name.



Our offline toolkit environment is now configured to speak to the central management server.

Since we already published the Inventory app in Lab 3, now all we need to do is publish the associated Product containing the APIs.

7. Click **Publish** button once more and select our target:

A screenshot of a "Publish" dialog box. At the top is a blue header bar with a "Publish" button. Below it is a section titled "Other Orgs" which contains a list of publishing details: Catalog: Sandbox (sb), App: Inventory (Inventory), Org: Sales (sales), and Server: mgr.think.ibm. This entire list is highlighted with a red rectangular border. At the bottom of the dialog is a link "Add and Manage Targets".

8. Check the box to **Stage or Publish products**:

Publish

A screenshot of a "Publish" options dialog. It contains four checkboxes:

- Publish application
- Stage or Publish products (this checkbox is highlighted with a red border)
- Stage only
- Select specific products



Here we have the opportunity to select what gets published. If we were working on multiple API products as part of this project, we could choose specific ones to publish.

Also, the option exists to only Stage the product. A Stage-only action implies that we'll push the configuration to the Management server, but not actually make it available for consumption yet. The reason for doing this may be because your user permissions only allow staging, or that a different group is in charge of publishing Products.

9. Click the `Publish` button to make the API available in the Developer Portal and enforced on our API Gateway.
10. Wait a moment while the Product is published, a `Success` message will appear letting you know the step is complete:


```
Success Successfully published products
```
11. Close the Firefox web browser.
12. In the `Terminal Emulator`, use the `control+c` keyboard command to quit the API Designer process.

Lab 6 - Validation

Using both the CLI and the API Designer, a lot of files have been created and updated. Before moving on to the next lab, a simple script has been provided for you which will validate your source files against those of a completed lab. These next steps will help ensure that easily made typing errors do not cause problems down the line.

1. From the `Terminal Emulator`, type:

```
validate_lab 6
```

2. The script will execute a series `diff` commands against specific files in your project folder (`~/ThinkIBM/inventory`)
3. If the output of the `validate_lab` script includes discrepancies, you may merge the corrected changes into your source files by typing:

```
merge_lab 6
```

4. If a merge was necessary, you will also need to re-publish the API Product in order to update the API Connect server with the latest changes.
5. Run the `apic edit` command to launch the API Designer, then repeat the steps from a moment ago to re-publish the product.

Lab 6 - Conclusion

Congratulations! You have completed Lab 6.

In this lab, you learned:

- About API Products
- How to publish an API Product

Proceed to [Lab 7 - Consumer Experience](#)

Lab 7 - Consumer Experience

In this lab, you will learn the consumer experience for APIs that have been exposed to your developer organization.

Using the developer portal, you will first login in as the admin user to load a Think IBM custom theme. You will then login as a developer to register your application and then subscribe to an API and test that API.

Lab 7 - Objective

In the this lab, you will learn:

- How to add a custom theme to change the overall look and feel of the developer portal.
- How to register an application with the developer portal.
- How to subscribe to a plan in order to consume an API.
- How to test an API from the developer portal.
- How to consume an API from a sample test application.

Lab 7 - Case Study Used in this Tutorial

In this tutorial, **ThinkIBM** wishes to brand their developer portal to their own look and feel. They also want to provide a self-service experience to application developers who are potential consumers of their APIs.

Lab 7 - Step by Step Lab Instructions

7.1 - Customize the Developer Portal

In this section, you will leverage a custom prebuilt theme. A theme file is packaged as a zip file that contains a set of predefined files. The theme zip file is what is loaded into the developer portal. The best method for creating a custom theme is to obtain an existing custom theme, rename it and change the files as explained in the table below to meet your customization needs.

| Filename | Description |
|----------|-------------|
| | |

overrides.css

This style sheet overrides the fonts, colors and other defaults from the inherited base IBM API Connect theme. The name of this css file is specified in the theme.info file.

screenshot.png

The file contains a screenshot of the home page for the custom theme and is used in the appearance settings so you can easily find the theme you want to enable and set as the default theme. When you are finishing completing the edits to the overrides.css and have set up the welcome banner to your satisfaction, you should take a screen shot of the developer portal home page and capture it with Snagit or some other screen capture tool and place the file in the top level directory of the theme file. The name of this screenshot file is specified in the theme.info file.

template.php

The template.php file contains your sub-theme's functions to manipulate Drupal's default markup. It is one of the most useful files when creating or modifying Drupal themes. With this file you can modify any theme hooks variables or add your own variables, using preprocess or process functions. Override any theme function: that is, replace a module's default theme function with one you write.

| | |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <p>Call hook alter functions, which allow you to alter various parts of Drupal's internals, including the render elements in forms. See http://api.drupal.org for more information about these functions.</p> |
| <code>thinkibm_connect_theme.info</code> | <p>The .info file is a static text file for defining and configuring a theme. Each line in the.info file is a key-value pair with the key on the left and the value on the right, with an "equals sign" between them (e.g. name = my_theme). The info file naming conventions needs to have the theme name specified as part of the filename. The project key in the .info file contains the name of the theme.</p> |
| <code>theme-settings.php</code> | <p>Drupal themes get a number of configurable settings options for free. For example, most provide toggle switches for the search box, site slogans, user pictures, and so on. Similarly, most provide file uploading widgets to add a custom logo or favicon. These settings are easy: Drupal will add them to the theme's configuration page by default, so it takes no extra work. We want to create our own custom setting, however; one that adds a hidden field that contains the current release</p> |

information to the Theme configuration form. To do that, we'll need to add a new file to the theme:

`theme-settings.php`. The function name specified within this file needs to be prefixed with the theme name: `thinkibm_connect_theme`.

`thinkibm_welcome_banner.png`

This file provides the image that shows up in the welcome banner. Although the welcome banner does not need to be part of the theme zip file contents, it is useful to place the welcome banner with other theme files.

`favicon.ico`

In web development, you can provide a small logo for your site that appears near the address bar and in the bookmarks folder in a visitor's browser. This logo is called the favicon. Drupal provides a default one, which is the recognizable water drop logo. Using the Drupal logo as the favicon is fine but if you really want to make your site stand out, you should provide your own. Favicon files are in the .ico format and are extremely small in dimensions. The default Drupal favicon is 32 pixels high by 32 pixels wide, many browsers use a 16 x 16 pixel version that can be included in the same file. This is because the favicon is only an icon that shows up in the

| | |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| | address bar and favorites (bookmarks) list and typically there is not a lot of room there. Any favicon that you create should be just as small. |
| logo.png | The default logo that appears at the top left-hand side of the developer portal page. |

7.1.1 - Install and Configure a Custom Theme

Now that we have explained the contents of a custom theme file, it is time to load the custom theme.

Note:* If this section is already complete in your lab environment, follow the steps and review the options.

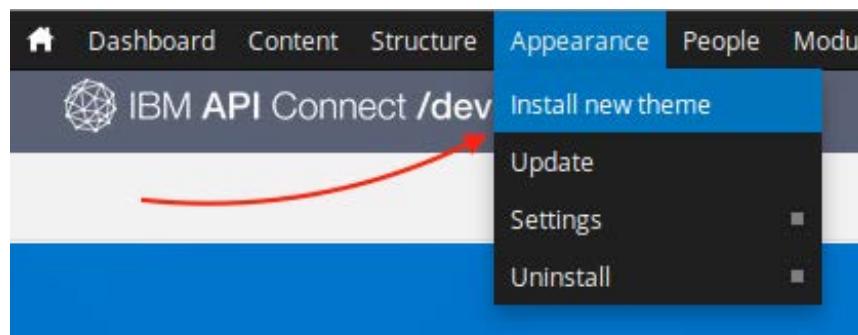
1. From the Firefox browser, click the `Portal` bookmark on the toolbar to lauch the developer portal.



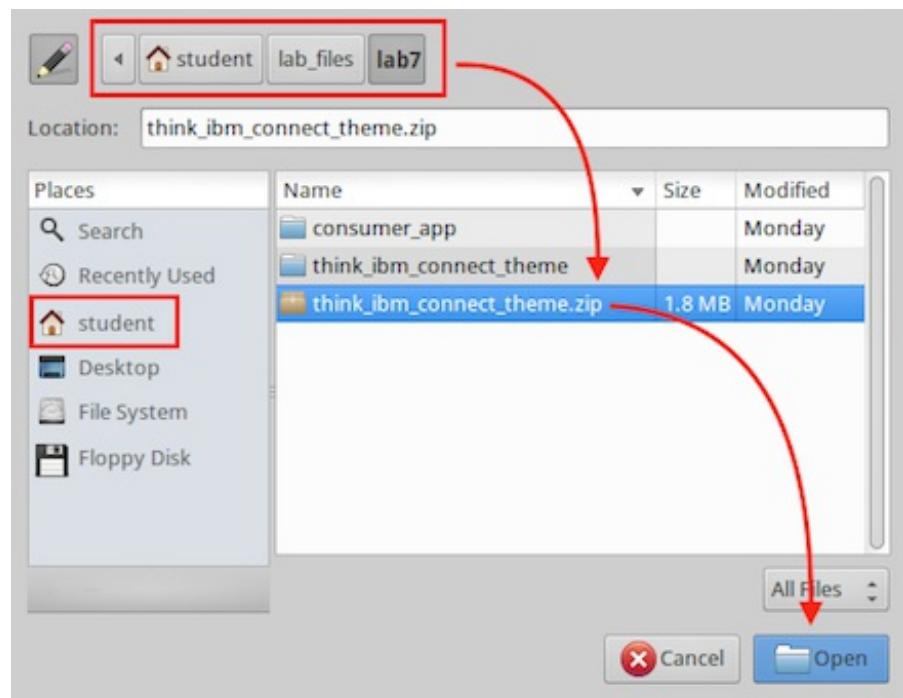
2. Login into the developer portal as an administrator using a username of `admin` and a password of `Passw0rd!`.

The screenshot shows the 'User login' page of the IBM API Connect developer portal. At the top, there are links for 'Create new account', 'Log in' (which is underlined), and 'Request new password'. Below these are fields for 'Username' (containing 'admin') and 'Password' (containing 'Passw0rd!'). Both fields have placeholder text below them. At the bottom is a red-bordered 'Log in' button.

3. From the Administrator menu, select `Appearance` and then `Install new theme`.



- Click the **Browse** button and navigate to the `/home/student/lab_files/lab7/` directory and select the `think_ibm_connect_theme.zip` file , then click the **Open** button.



- Click the **Install** button to install the **ThinkIBM** custom theme.

You can find [modules](#) and [themes](#) on [drupal.org](#). The following file extensions are supported: `zip tar tgz gz bz2`.

Install from a URL

For example: `http://ftp.drupal.org/files/projects/name.tar.gz`

Or

Upload a module or theme archive to install

 think_ibm_connect_theme.zip

For example: `name.tar.gz` from your local computer

- The installation of the custom theme completes. Click the **Enable newly added themes** link.
- Scroll down the list of themes to find the `thinkibm_connect 7.43` theme. Click the **Enable and set default** link link.



thinkibm_connect 7.43

A sample theme designed for IBM API Connect but otherwise it is a minimal, re-colorable theme for Drupal 7 that uses HTML5 and [Adaptivetheme 7.x-3.x](#) base theme.

[Enable](#) [Enable and set default](#)

8. We need to go into settings for the theme and save the configuration.
Click the [Settings](#) link under the theme.

thinkibm_connect 7.43 (default theme)

A sample theme designed for IBM API Connect but otherwise it is a minimal, re-colorable theme for Drupal 7 that uses HTML5 and [Adaptivetheme 7.x-3.x](#) base theme.

[Settings](#)

9. Scroll down to the bottom of the settings page to find the **Toggle Display** section. Expand the **Toggle** click the **Save Configuration** button at the bottom of the screen:

- - Toggle display

Enable or disable the display of certain page elements.

Logo

Site name

Site slogan

User pictures in posts

User pictures in comments

User verification status in comments

Shortcut icon

- - Logo image settings

- - Shortcut icon settings

- - jQuery Update

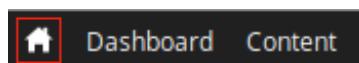
You can optionally select a different version of jQuery to use for pages that are rendered using this theme. This is useful for administrative based themes.

Theme specific jQuery version

Site default (1.8) ▾

[Save configuration](#)

10. Click the [Home](#) icon to return to the home page.



7.1.2 - Customize the Welcome Banner

Note:* If this section is already complete in your lab environment, follow the steps and review the options.

Now we want to change the default Welcome banner to use our custom Welcome banner image.

1. Navigate to the blocks content page by selecting `Content > Blocks` from the admin menu.



2. Underneath operations select `Edit` to the right of the Welcome banner block.

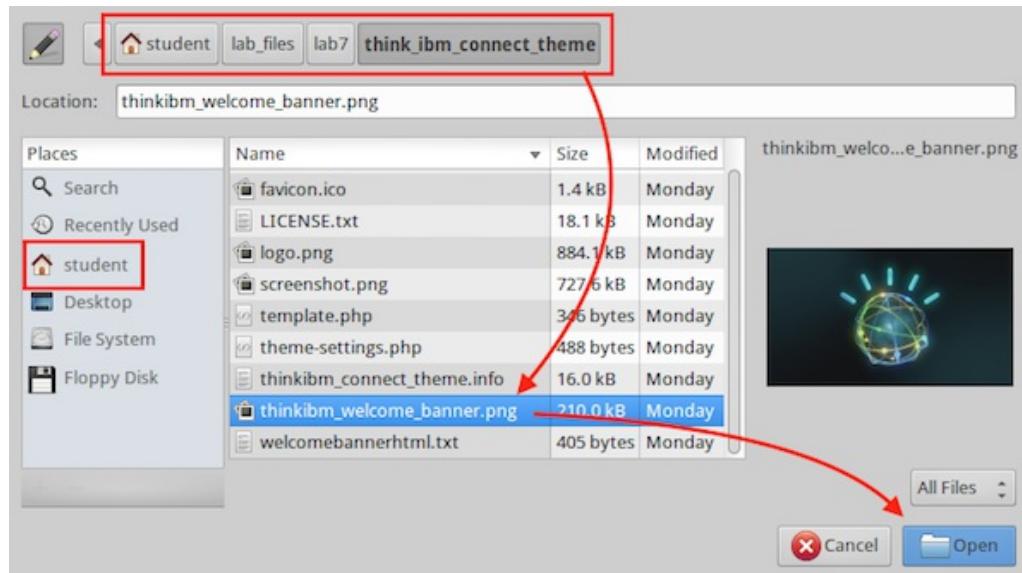
A screenshot of the 'Edit Block' page for the 'Welcome Banner' block. The top navigation bar shows 'Home > Administration > Content'. The main area has a 'Filters' section with a dropdown menu showing 'Banner' selected. Below is a table listing blocks:

| Title | Type | Operations |
|----------------|----------------------|---------------------------------------------|
| Welcome Banner | banner_block | Edit Delete |
| 11 | popularproduct_block | Edit Delete |
| socialblock | socialblock_block | Edit Delete |

3. Scroll down the banner edit page, click the `Browse` button underneath **Image** to launch the file explorer.

4. Navigate to the

`/home/student/lab_files/lab7/think_ibm_connect_theme` directory and select the `thinkibm_welcome_banner.png` file. Then, click the `Open` button.



5. Click the **Upload** button to upload the new welcome banner image.
6. Scroll to the bottom of the page and click the **Save** button.

The screenshot shows the configuration interface for a content block. At the top, there is a preview of the uploaded image 'thinkibm_welcome_banner.png' (205.1 KB). To its right is a 'Remove' button. Below the image, there is a 'Background Image' section. Underneath the image preview, there are two dropdown menus for 'View Mode': 'Default' and 'Edit the view mode of the Bean'. There are also two checked checkboxes: 'Create new revision' and 'Set Revision as default'. A 'Log message' input field is present with a placeholder: 'Provide an explanation of the changes you are making. This will help other authors understand your motivations.' At the bottom of the form are two buttons: 'Save' (highlighted with a red box) and 'Delete'.

7. Close the content block settings.

The screenshot shows the 'Content' administration interface with the 'Blocks' tab selected. At the top, there are tabs for 'Content', 'Blocks', and 'Comments'. Below the tabs, there's a breadcrumb navigation: 'Home > Administration > Content'. A red box highlights the 'Blocks' tab. On the left, there's a sidebar with an 'Add Block' button and a 'Filters' section containing a dropdown menu with options: 'Banner', 'Popular API', 'Popular Product', and 'Social Block'. Below the filters is a 'Type' dropdown set to 'socialblock_block', an 'Items Per Page' dropdown set to 50, and buttons for 'Filter' and 'Reset'. The main area displays a table with three rows:

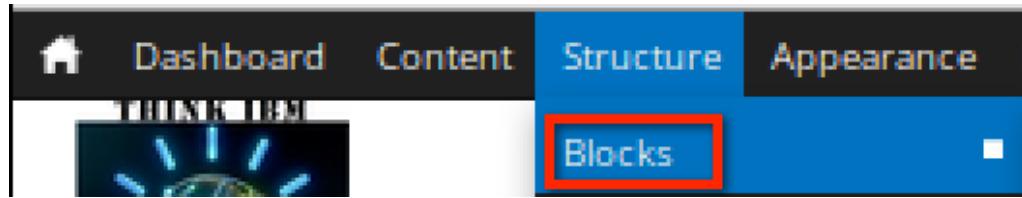
| Title | Type | Operations |
|----------------|----------------------|-------------|
| Welcome Banner | banner_block | Edit Delete |
| 11 | popularproduct_block | Edit Delete |
| socialblock | socialblock_block | Edit Delete |

7.1.3 - Change the Region Settings for the Content Blocks

Note:* If this section is already complete in your lab environment, follow the steps and review the options.

The region settings for some of the content blocks have been reset, so we will set these back.

1. Navigate to the region settings for block content page by selecting **Structure > Blocks** from the admin menu.



2. Locate the **Sidebar first** block. You will see entries for **Navigation**, **Support** and **User login**.
3. For **both** the **Navigation** and **User Login** entries, select **- None** from the drop down menu.

The screenshot shows the configuration page for the 'Sidebar first' block. At the top, there are buttons for 'Add block' and 'Add taxonomy menu block'. Below that is a table with two columns: 'Block' and 'Region'. The 'Block' column contains the entry 'Sidebar first'. The 'Region' column contains a dropdown menu with several options: 'Sidebar first', '- None -', 'Sidebar second', 'Top menu bar', 'Header', 'Menu Bar', and '...'. A red arrow points from the 'Navigation' entry in the 'Sidebar first' block list to the open dropdown menu in the 'Region' column.

| Block | Region |
|---------------|------------------------------------------------------------------------------------------|
| Sidebar first | Sidebar first - None - Sidebar second Top menu bar Header Menu Bar ... |

4. The only entry left in the section should be **Support**.

Block

Region

Sidebar first

+ Support Sidebar first

Sidebar second

No blocks in this region

5. Scroll down to the bottom of the screen and click the **Save Blocks** button.

+ User login [- None -] configure

+ socialblock [- None -] configure

Save blocks

6. Click close on the block region settings.

Content

Home » Administration » Content

Add Block

Filters

| Type | Title | Type | Operations |
|-----------------|----------------|----------------------|-------------|
| Banner | Welcome Banner | banner_block | Edit Delete |
| Popular API | 11 | popularproduct_block | Edit Delete |
| Popular Product | | | |
| Social Block | socialblock | socialblock_block | Edit Delete |

7. You are finished with customizing the developer portal. There is a lot more that can be customized than what we have time for in this lab. Log out of the developer portal.



7.2 - Register an Application as a Developer

In this section, you will log into the portal as a user in the application developer role, then register an application that will be used to consume APIs.

1. Login into the developer portal as an application developer using a username of `developer@consumer.ibm` and a password of `Passw0rd!`.



Create an account

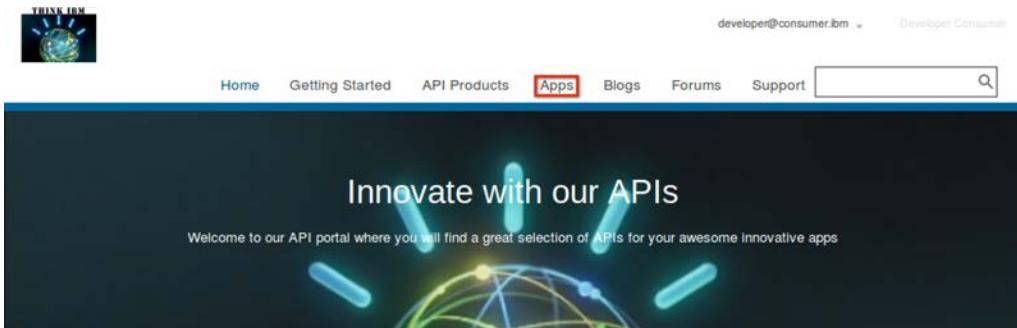
Home Getting Started API Products Blogs Forums Support

[Create new account](#) [Log in](#) [Request new password](#)

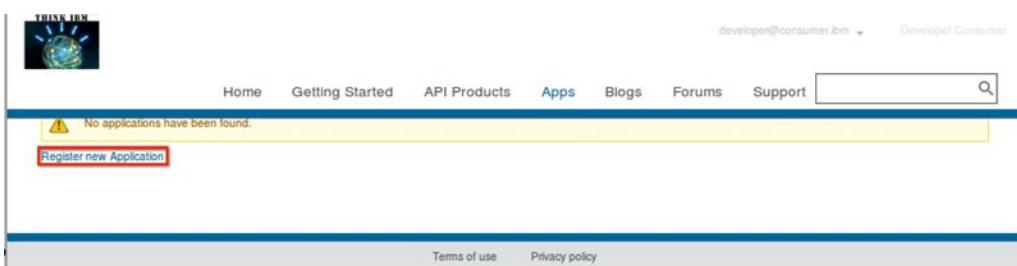
Username * Enter your developer think.ibm username.

Password * Enter the password that accompanies your username.

2. Click the **Apps** link.



3. Click the **Register new Application** link.



4. Enter a title and description for the application as shown below and click the **Submit** button.

developer@consumer.ibm + Developer Consumer

Home Getting Started API Products Apps Blogs Forums Support

Register application

Title *

Description

OAuth Redirect URI

The URL authenticated OAuth flows for this application should be redirected to:

5. We need to capture the Client Secret and Client ID in a text editor for later use by our web application. Select the **Show Client Secret** checkbox next to Client Secret at the top of the page and the **Show** checkbox next to Client ID.

• Application created successfully.
• Your client secret is:
`mX4qQ8aY8dT6rD7gW4kK6uA0wG8pL5uD2dO8jM2pR2mD0dH7IS`

Show Client Secret

< Back to Apps

Think IBM Web Consumer

2016-04-11

Description
Think IBM Web Consumer that will access APIs to view inventory items and comment on inventory items.

Client Credentials | [+](#)

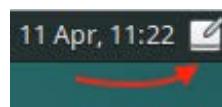
Client ID
`2bfbbe5a9-3eba-4a20-9713-73d0261adb77`

Show [Reset](#)

Client Secret

[Verify](#) [Reset](#)

6. Launch the **Notes** application as shown below.



7. Navigate back to the browser and copy/paste both the Client ID and Client Secret into the Notes application window as shown below. Add a notation above each item so you know which value is the Client ID and which is the Client Secret. Do not exit the Notes application as we will need these values later in this lab.

• Application created successfully.
• Your client secret is:
`mX4qQ8aY8dT6rD7gW4kK6uA0wG8pL5uD2dO8jM2pR2mD0dH7IS`

Show Client Secret

< Back to Apps

Think IBM Web Consumer

2016-04-11

Description
Think IBM Web Consumer that will access APIs to view inventory items and comment on inventory items.

Client Credentials | [+](#)

Client ID
`2bfbbe5a9-3eba-4a20-9713-73d0261adb77`

Show [Reset](#)

Client Secret
`mX4qQ8aY8dT6rD7gW4kK6uA0wG8pL5uD2dO8jM2pR2mD0dH7IS`

Notes - Notes

Client Secret
`mX4qQ8aY8dT6rD7gW4kK6uA0wG8pL5uD2dO8jM2pR2mD0dH7IS`

Client ID
`2bfbbe5a9-3eba-4a20-9713-73d0261adb77`

7.3 - Subscribe to a Plan for the ThinkIBM APIs

In this section, we will subscribe to a plan for the ThinkIBM APIs using the Think IBM Web Consumer application.

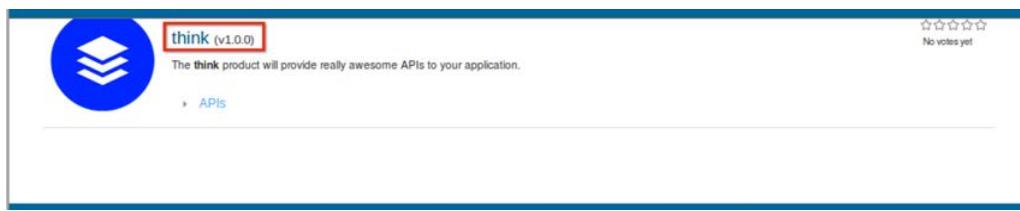
1. Click the **API Products** link.

THINK IBM

developer@consumer.ibm - Developer Consumer

Home Getting Started **API Products** Apps Blogs Forums Support

2. Click the `think (v1.0.0)` API product link.



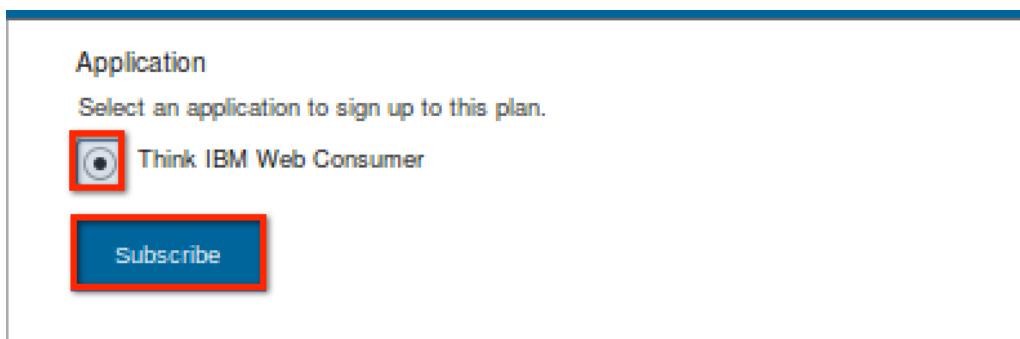
3. Click the `Plans` link on the left-hand navigation menu and then click the `Subscribe` button under the **Silver** plan as shown below.

A screenshot of a 'Plans' page for the 'think 1.0.0' API product. On the left is a sidebar with 'APIs' and a 'Plans' link, which is highlighted with a red box. The main area shows a table of plans for four different APIs: 'inventory 1.0.0', 'financing 1.0.0', 'logistics 1.0.0', and 'oauth 1.0.0'. Each row has two columns: 'Gold' (locked) and 'Silver'. The 'Silver' column for all APIs shows a price of '100 per hour'. At the bottom of the table are two 'Subscribe' buttons, with the one under the 'Silver' column for 'logistics 1.0.0' highlighted with a red box. There is also an 'i' icon with a blue border.

Alt text

The Gold plan requires approval by the API provider for any subscription requests and allows unlimited requests for a given time period. The Silver plan is limited to 100 requests per hour and does not require approval by the API provider for subscription requests.

4. Select the `Think IBM Web Consumer` toggle and click the `Subscribe` button.



5. The `Think IBM Web Consumer` Web application is now subscribed to the Silver plan for the `think` API product.



think (v1.0.0)

The think product will provide really awesome APIs to your application.

▶ APIs

6. Let's validate that we subscribed to the Silver plan for the think v1.0.0 API product. Click the Apps link.

A screenshot of the IBM API Portal homepage. At the top, there is a navigation bar with links for Home, Getting Started, API Products, Apps (which is highlighted with a red box), Blogs, Forums, and Support. A search bar is also present. The main content area features a dark background with a globe and glowing blue lines, and the text "Innovate with our APIs". Below this, a sub-section says "Welcome to our API portal where you will find a great selection of APIs for your awesome innovative apps".

7. Click the Think IBM Web Consumer application link.

A screenshot of the application details page for "Think IBM Web Consumer". The page shows a brown circular icon with a smartphone and gear icon, the application name "Think IBM Web Consumer" (with a red box around it), and the creation date "2016-04-07". A description below states: "Think IBM Web Consumer application that will access APIs to view inventory items and comment on the items." There is also an "Update" button.

8. The Think IBM Web Consumer application is subscribed to the Silver plan for the think product as shown below.

A screenshot of the application settings and subscriptions page for "Think IBM Web Consumer". The top section shows the application details again. Below that is a "Client Credentials" section with fields for "Client ID" and "Client Secret", each with a "Show" checkbox and a "Reset" button. The "Subscriptions" section shows a list of products and plans: "think (v1.0.0) silver" (with a red box around it), "inventory (v1.0.0)", "financing (v1.0.0)", and "logistics (v1.0.0)". An "Unsubscribe" button is located next to the selected subscription.

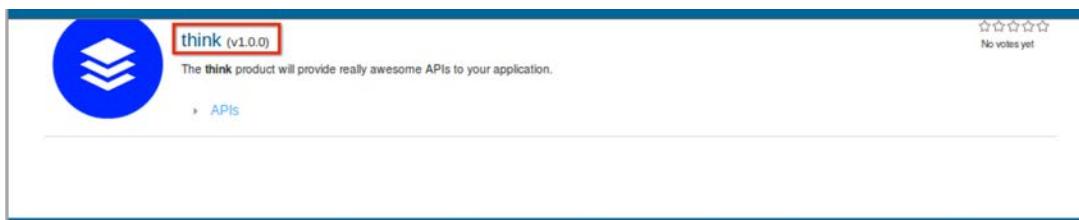
7.4 - Test think Product APIs from the Developer Portal

In this section, we will use the developer portal to test one of the think product APIs. This is useful for application developers to **try out** the APIs before their application is fully developed or to simply see the expected response based on inputs they provide the API. Later in this lab, you will run an actual application that is already developed and uses these same APIs. We will test the `logistics` API from the developer portal.

1. Click the `API Products` link.



2. Click the `think (v1.0.0)` API product link.



1. Click the `logistics 1.0.0` link on the left-hand navigation menu and then expand the `GET /shipping` path by clicking on the twisty next to the path.

A screenshot of the 'logistics 1.0.0' API details page. The left sidebar shows 'Paths' with 'GET /shipping' selected. The main panel shows the 'Paths' table with one row for '/shipping'. The 'Example Request' section contains a curl command. The 'Example Response' section shows a JSON object with 'xyz' and 'next_day' fields.

2. Scroll down to the **Try this operation** section for the `GET /shipping` path. Enter any zip code and click the `Call Operation` button.

The screenshot shows the API Connect interface. On the left, there's a sidebar with 'think 1.0.0' at the top, followed by a list of APIs: inventory 1.0.0, financing 1.0.0, logistics 1.0.0, oauth 1.0.0, Security, Operations, GET /authorize, POST /authorize, POST /token, Definitions, and /stores. The 'Security' section is currently selected. In the center, there's a 'shipping' section with a '200 200 OK' status. To the right, the 'Try this operation' panel is open, showing the URL <https://gateway.pot.ibm/sales/sb/logistics/shipping>. It has fields for 'Client ID' (set to 'Think IBM Web Consumer'), 'Headers' (content-type: application/json, accept: application/json), and 'Parameters' (zip: 60601). A red box highlights the 'Call operation' button at the bottom.

3. Scroll down below the `Call operation` button. You should see a `200 OK` and a response body as shown below.

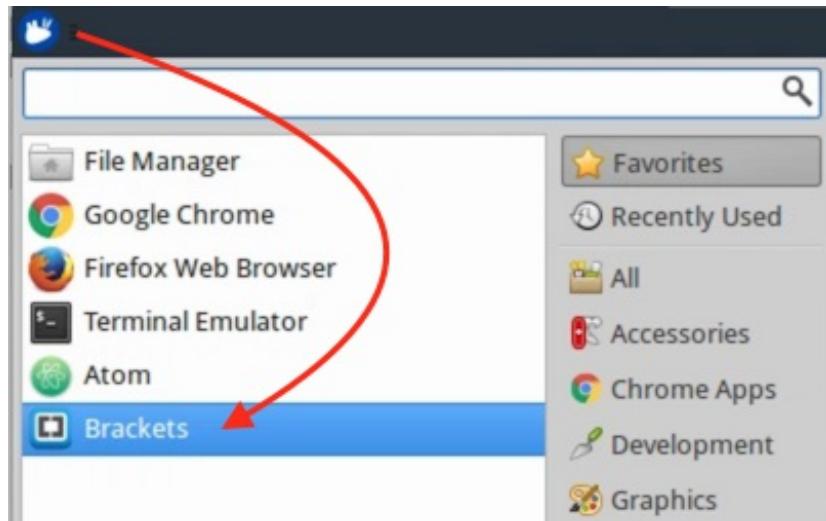
The screenshot shows the 'Response' panel. At the top, it says '200 OK'. Below that is the response header:
Content-Type: application/json
X-Global-Transaction-ID: 372256
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 99
The response body is a JSON object:
{
 "xyz": {
 "next_day": "31.52",
 "two_day": "21.57",
 "ground": "16.59"
 },
 "cek": {
 "next_day": "28.65",
 "two_day": "19.61",
 "ground": "15.08"
 }
}
A red box highlights the entire JSON response body.

7.5 - Configure and Run a Consumer Application

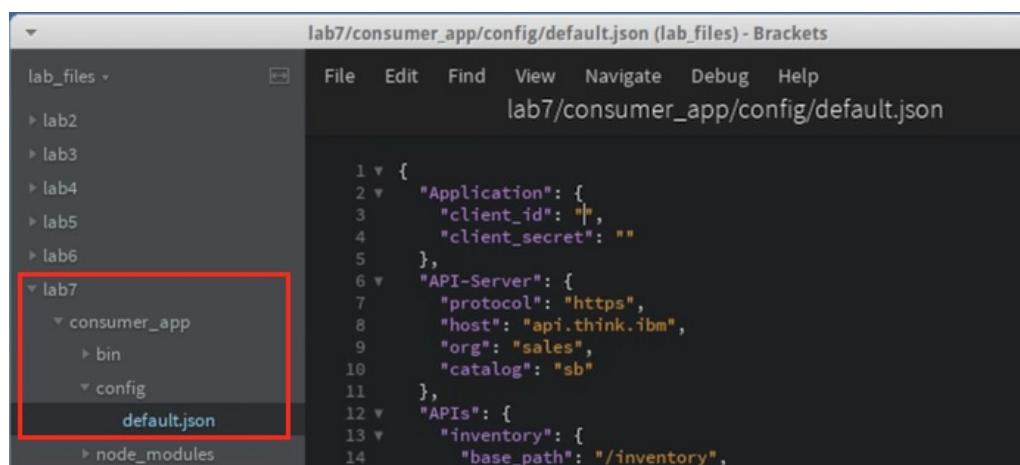
In this section, we will edit a config file for an implementation of the `Thin k IBM Web Consumer` application. The application is pre-configured to communicate with our API Connect environment and the API's which you just created.

The application will handle the OAuth token exchange on our behalf, as well as provide the Client ID and Client Secret for all API calls so that they will pass entitlement validation.

1. Launch the Brackets text editor window from the task bar if it is already open, or from the favorites menu as shown below:



2. In the Brackets text editor file tree menu, expand the lab7 > consumer_app > config folder and select the default.js file to edit the source.

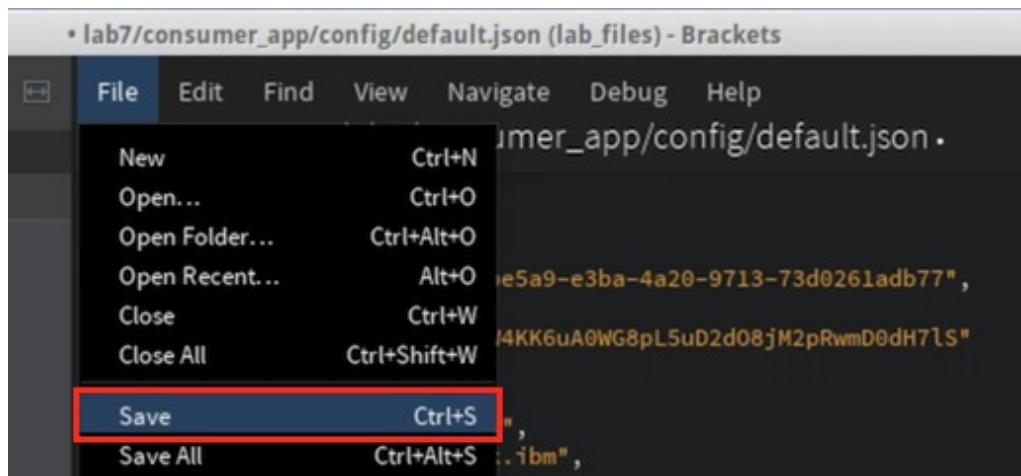


3. Using the Client ID and Client Secret values in the Notes application, copy/paste them into the "client_id": "" and "client_secret": "" values between the quotes in the default.json file as shown below:

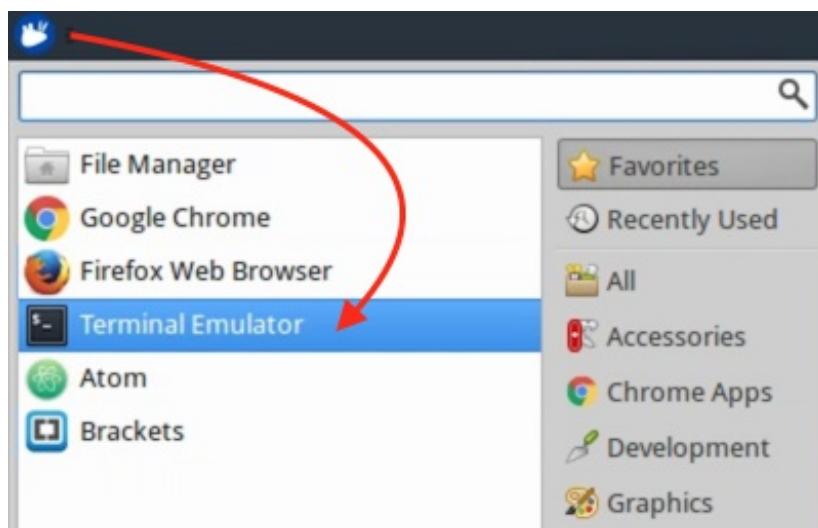
A screenshot of the Brackets text editor showing the modified default.json file. The "client_id" and "client_secret" values are highlighted with a red box:

```
1 ▼ {  
2 ▼   "Application": {  
3     "client_id": "2bfbe5a9-e3ba-4a20-9713-73d0261adb77",  
4     "client_secret":  
5       "mX4qQ8aY8dT6rD7gW4KK6uA0WG8pL5uD2d08jM2pRwmD0dH7LS"  
6   },  
7 }
```

4. With the default.json tab in the Brackets text editor window selected, choose File > Save from the application menu.



5. Now we will open a fresh Terminal Emulator window so we can start the Think IBM Web Consumer application. Click the favorites menu and select Terminal Emulator as shown below.



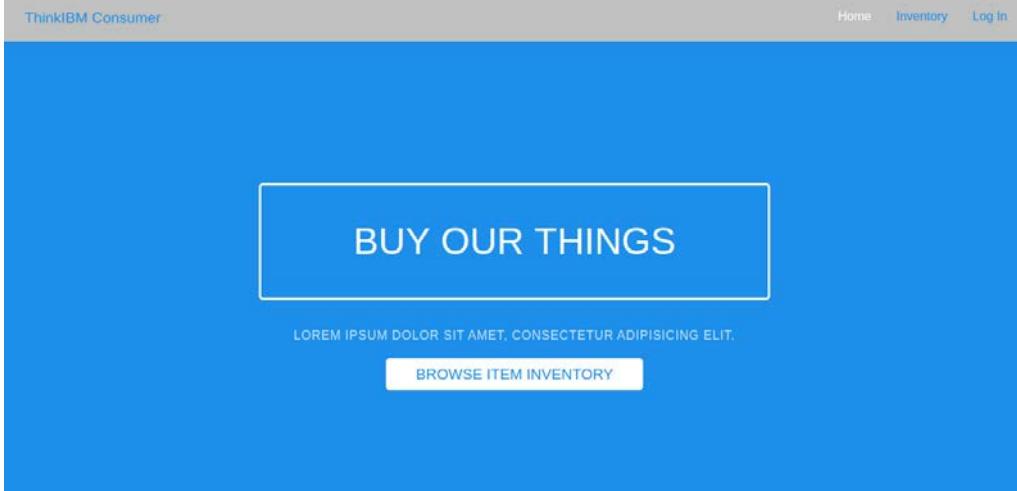
6. Navigate to the /home/student/lab_files/lab7/consumer_app/ directory by entering the cd command as shown below:

```
cd ~/lab_files/lab7/consumer_app/
```

7. Start the Think IBM Web Consumer application by entering the npm start command as shown below. The application is a **Node.js** application.

```
npm start
```

8. The Think IBM Web Consumer application launches in a browser window.



7.6 - Test the APIs from the Think IBM Web Consumer Application

1. Recall that in Lab 4 you secured the `inventory` API by requiring OAuth. If you attempt to view the Item Inventory before logging in, you will be challenged for a username and password. Either click on the `Browse Item Inventory` button, or the `Log In` link to open the login prompt.
2. Enter any Username and Password then click the `Log In` button. Our sample authentication service is a dummy repository that will accept any credentials provided.

A screenshot of a login form. It consists of two input fields: one for "Username" containing the text "john" and another for "Password" containing five dots ("....."). Below the inputs is a large blue "Log In" button. To the right of the button is a small icon of a person with an exclamation mark inside a square frame.

The consumer app will contact our OAuth Token URL and handle the token exchange using standard OAuth procedures. If you're interested in seeing the token, you can switch over to the terminal from where you launched the consumer app and view the logs. Look for a line similar to:

Using OAuth Token: AAEKZGEwZDgxNDMtMzIxMS00ZDgyLWE3MGYtMTJmY2UyYjk1YmUyxzqSMdEr7wM8XU_ed03dmxGdmC1ZGa9UC9Ibh-r0VNlflzd6blfDq4CGaNsD1qn-KESw6Q6RN_TPA0IfMdIn1nHY4ZpGRYa5N0f7mgY2Jg4Tfhm0IlhCUq5HRvoo7c4SIX7SAS3rL998_BvMVBST_g

3. Once you are logged in, the consumer application will redirect you to the item inventory page. The data displayed on the page is being powered by our `inventory` API! The consumer application is calling to our API, which is then being sent to our LoopBack application which handles the connection to the MySQL data source where the inventory data is persisted.
4. Click around the pages. Use the review form to leave new reviews for items. Recall how the item reviews are separate data models stored in MongoDB, yet are related to the items through our LoopBack application.

Notice how the product rating is updated automatically as new reviews are posted. This happens because of the custom code you added to the LoopBack application in Lab 3.

Customer Reviews (2)

★★★★★ by Bob on Mon Apr 11 2016

"Very cool!"

★★★★★ by Jim on Mon Apr 11 2016

"Way overpriced!"



Dayton Meat Chopper



\$4599.99

[Calculate Monthly Payment](#)

5. Continue testing the features of the consumer application. Try entering your home zip code to obtain shipping data. This feature is

powered by the `logistics` APIs that were built using advanced assembly techniques in Lab 5. Your quotes will vary based on the zip code provided.

The results also provides a link to the nearest store.

Calculate Shipping

| | | |
|-------------------------------------------|----------|---------|
| XYZ | Next Day | \$34.75 |
| | Two Day | \$23.78 |
| | Ground | \$18.29 |
| <hr/> | | |
| CEK | Next Day | \$31.59 |
| | Two Day | \$21.61 |
| | Ground | \$16.63 |
| <hr/> | | |
| Pickup from nearest store | | |

- Finally, try clicking on the `Calculate Monthly Payment` link. This feature is powered by the `financing` API that was created using our REST-to-SOAP assembly in Lab 5.

Dayton Meat Chopper



\$4599.99

Monthly Payment: \$199.55 at 3.9% for 24 months

- Repeat the steps on a few of products to drive data into the analytics database.

Lab 7 - Completion

Congratulations! You have completed Lab 7!

Appendix A - Troubleshoot the inventory API

In this appendix, you review and apply troubleshooting steps for the case study in the lab exercises.

Apply the steps to verify your copy of the case study, and correct the issues that you identify.

Appendix A - Objective

In this lab, you will learn:

- How to verify the key components in the exercise case study
- How to troubleshoot and correct common issues with the case study
- How to isolate implementation issues in your own workstation

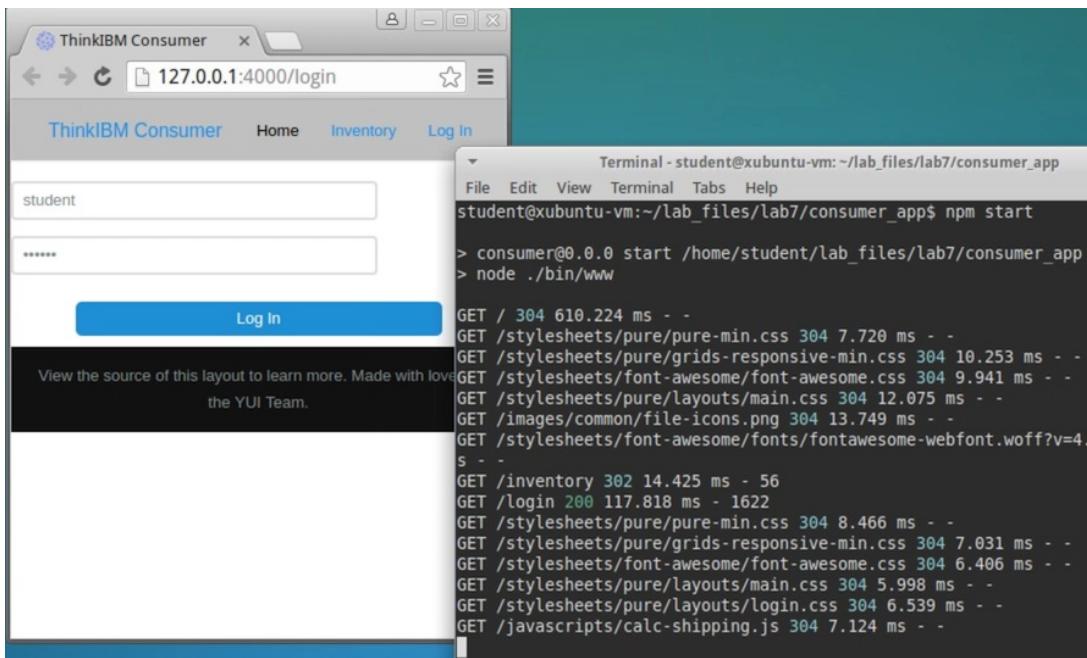
Appendix A - Case Study Used in this Tutorial

After you finished exercise 6, you completed the *ThinkIBM* inventory, logistics, financing, and OAuth API implementations. In this appendix, you verify the API gateway and API implementation in your lab environment. You troubleshoot and correct common issues with the exercises.

Appendix A - Step by Step Troubleshooting Instructions

Appendix A.1 - Review the inventory items API operation

In exercise section 7.3, you test the inventory items API operation that you built from exercise 2 to exercise 6. If your copy of the `consumer_app` returns a runtime error, you must isolate and determine which part of the inventory API caused the issue.



The `consumer_app` in lab 7 makes three API calls when you select `log in`:

- Call #1: Consumer app --> API Gateway `/oauth20/authorize`
- Call #2: Consumer app --> API Gateway `/oauth20/token`
- Call #3: Consumer app --> API Gateway `/inventory/items` --> IBM HTTP Server --> Collective plug-in --> Collective server --> Inventory LoopBack app

In *call #1*, the consumer application calls the OAuth 2.0 provider's `authorize` API operation. The `authorize` operation redirects your web browser to a log in page. You enter any value for the user name and password, and the authorization URL returns HTTP status code 200 to the OAuth 2.0 Provider. In turn, the `authorize` API operation returns an OAuth *authorization bearer token* to the consumer application.

In *call #2*, the consumer application sends the *authorization bearer token* to the `token` API operation, and a request to the `/inventory/items` resource. After the `token` API operation verifies that the bearer token is valid, it returns an OAuth 2.0 *access token* to the consumer application.

In *call #3*, the consumer application calls the `/inventory/items` API operation with the *access token*.

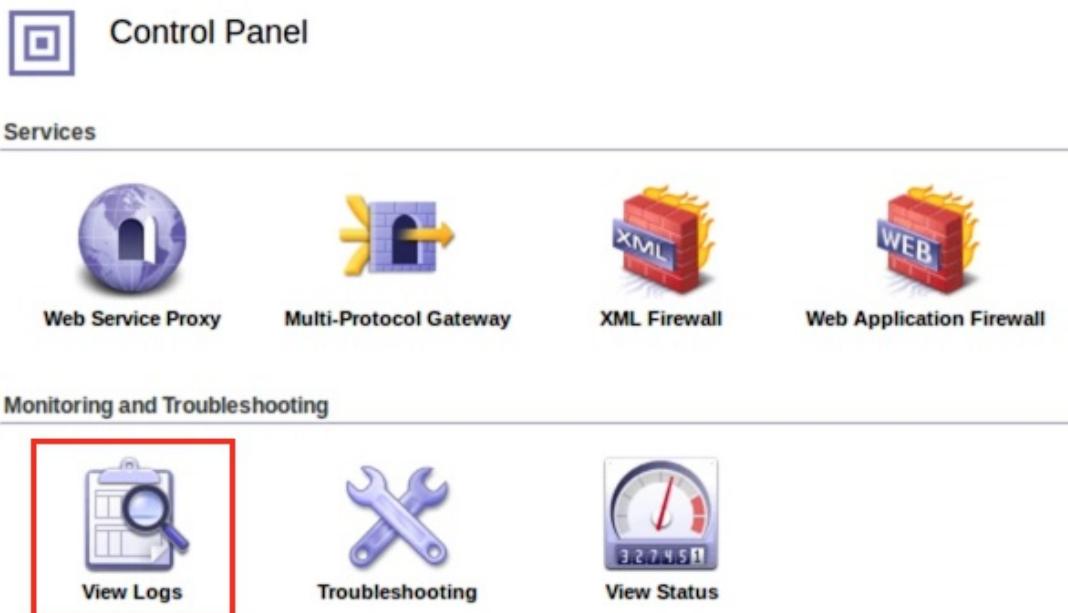
Appendix A.2 - Check whether the OAuth 2.0 `/oauth20/authorize` API operation call completed successfully

If you see the message `Using OAuth Token: ...` in your terminal console after you submitted your user name and password, your application completed *call #1* and *#2* successfully.

```
Using OAuth Token:  
AAEkM2M0MjdiNDAtMThkNS00NzkyLWFhNTUtOTc0NGNjZTAzNTdh-7Z1GFMi0  
HTY0FVGbxK_zX0c8Hr5tXd91zmrlTqV6LoaWlawbqw0h4b-PiXYf3Q0-TWtF0  
Dyp2cbjgvxT5kJnA
```

If you do not see this message in your terminal console, log into the DataPower web interface. Verify that there are no authorization errors at the Gateway before you continue to section A.3.

1. To open the DataPower web interface, open <https://dp.think.ibm:9090/> in a web browser.
2. Log in with the following credentials:
 - User name: `admin`
 - Password: `Passw0rd!`
 - Domain: `default`
3. In the control panel, select `view logs`.



Appendix A.3 - Verify that the consumer application sent an OAuth token in the /inventory/items API call

If *call #3*, the call from the consumer app to the `/inventory/items` API

operation fails, you see this error:

```
Inventory API call failed:
```

Complete these steps to print out the HTTP response message from the `/inventory/items` API call.

1. Examine the console log in your terminal after the sign-in page.
2. Review the HTTP request call to the `/inventory/items` API operation. Confirm that the call includes an `Authorization: Bearer` HTTP header.

MY OPTIONS:

```
{"method":"GET","url":"https://api.think.ibm/sales/sb/inventory/items?filter[order]=name%20ASC","strictSSL":false,"headers":{"X-IBM-Client-Id":"3c427b40-18d5-4792-aa55-9744cce0357a","X-IBM-Client-Secret":"X4vK7cY6xJ5tC0vR1qV5nD2jS1tL0tS1gU7dP3bQ7aY2fD4iE2","Authorization":"Bearer AAEkM2M0MjdiNDAtMThkNS00NzkyLWFhNTUtOTc0NGNjZTAzNTdh-7Z1GFMi0HTY0FVGbxK_zX0c8Hr5tXd91zmrlTqV6LoaWlawbqw0h4b-PiXYf3Q0-TWtF0Dyp2cbjgvxT5kJnA"}}}
```

3. Review the error message with the text

`Inventory API call failed:` . Specifically, examine the **response** section of the message.

```
Inventory API call failed:
```

```
{"name":"StatusCodeError","statusCode":404,"request":{"method":"GET","url":"https://api.think.ibm/sales/sb/inventory/items?filter[order]=name%20ASC","strictSSL":false,"headers":{"X-IBM-Client-Id":"3c427b40-18d5-4792-aa55-9744cce0357a","X-IBM-Client-Secret":"X4vK7cY6xJ5tC0vR1qV5nD2jS1tL0tS1gU7dP3bQ7aY2fD4iE2","Authorization":"Bearer AAEkM2M0MjdiNDAtMThkNS00NzkyLWFhNTUtOTc0NGNjZTAzNTdh-7Z1GFMi0HTY0FVGbxK_zX0c8Hr5tXd91zmrlTqV6LoaWlawbqw0h4b-PiXYf3Q0-TWtF0Dyp2cbjgvxT5kJnA"}}, "response": "<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">\n<html><head>\n<title>404 Not Found</title>\n</head><body>\n<h1>Not Found</h1>\n<p>The requested URL /inventory/items was not found on this server.</p>\n<hr>\n<address>IBM_HTTP_Server at inventory-576d5dc5e4b05a78b037514b-1466785559733.sales Port 80</address>\n</body></html>\n"}}
```

In this specific example, the Collective plug-in cannot invoke the

`/inventory/items` URL. This error occurs while the API gateway attempts to invoke the `inventory` API that you published in exercise 3.

The next step is to determine whether this is an issue with the `app-id` routing, the LoopBack API implementation, the collective member, or the collective plug-in routing paths.

Appendix A.4 - Verify the app-id routing policy in the inventory API definition file.

In exercise section 4.4, you defined the `app-id` application property in the `inventory` API definition. The API gateway invokes the collective member that matches the `app-id` value. In this section, confirm that the `app-id` value matches the identifier for the application that you published at the end of exercise 3.

1. Open `~/ThinkIBM/inventory/definitions/inventory.yaml` document in a text editor.

```
cd ~/ThinkIBM/inventory/definitions  
mousepad inventory.yaml &
```

2. Locate the `app-id` property in the `x-ibm-configuration` section of the `inventory.yaml` file.

```
Sandbox:  
properties:  
  app-server: 'http://appsvr.think.ibm'  
  app-id: inventory-576d5dc5e4b05a78b037514b-1466785559733.sa  
les
```

3. Confirm that the `app-id` value matches the **host header** value that you saved at the end of exercise 3.

```

inventory.yaml - Mousepad
File Edit View Text Document Navigation Help
x-ibm-configuration:
  testable: true
  enforced: true
  cors:
    enabled: true
  gateway: datapower-gateway
catalogs:
  apic-dev:
    properties:
      runtime-url: ${TARGET_URL}
  sb:
    properties:
      runtime-url: 'http://localhost:4001'
Sandbox:
  properties:
    app-server: 'http://appsvr.think.ibm'
    app-id: inventory-576d5dc5e4b05a78b037514b-1466785559733.sales

```

Notes - Notes

Host header:
inventory-576d5dc5e4b05a78b037514b-1466785559733.sales

If the `app-id` value does not match the host header value, you must correct this issue before you continue to section A.5.

1. Correct the `app-id` value in `inventory.yaml` to match the **host header** value.
2. Save your changes in the text editor.
3. From the terminal, launch the API Designer web application.

```

cd ~/ThinkIBM/inventory/
apic edit

```

4. Complete **Exercise 6.3, Publishing the API Product** to publish the changes in the `inventory.yaml` API definition to the API Management server.

Appendix A.5 - Call and verify the /inventory/items API operation directly

The next component to check is whether the Inventory LoopBack app that you published in lab 3 is running.

- Call #3: Consumer app --> API Gateway `/inventory/items` --> IBM HTTP Server --> Collective plug-in --> Collective server --> **Inventory LoopBack app**

The files that represent the collective server is stored in the `~/wlpn/` directory, with the following naming convention: `inventory-xyz-1` , where `xyz` is the same value that you copied for the host header ID. The first member in the cluster has a suffix of `-1` : for example, `inventory-576d5dc5e4b05a78b037514b-1466785559733-1` .

1. Examine the server configuration for the collective member that hosts the inventory application.

```
cd ~/wlpn/
ls
cd ~/wlpn/inventory-576d5dc5e4b05a78b037514b-1466785559733-1
/
cat join.json
```

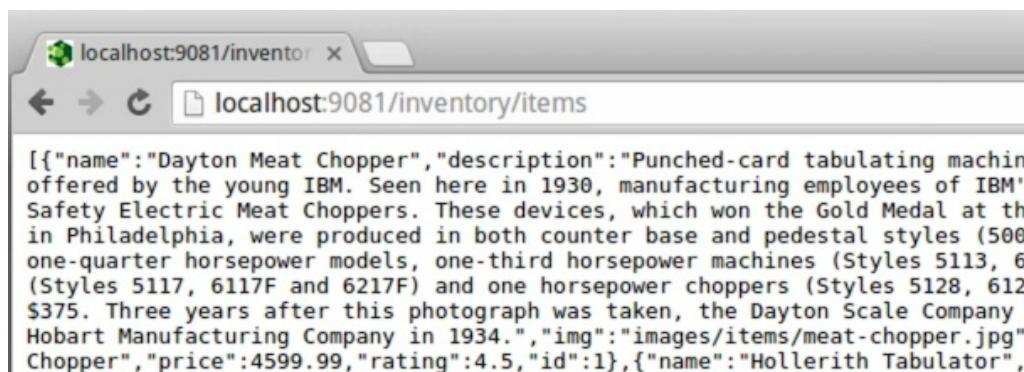
Note: Replace the collective server name with the one in your `~/wlpn` directory.

2. Write down the server port number from the `join.json` configuration file. For example, port `9081`.

```
    "clusterName": "apic.inventory-576d5dc5e4b05a78b037514b-146  
6785559733",  
    "appPort": "9081",  
    "adminPort": "9444"
```

3. Open a web browser to <http://localhost:9081/inventory/items>

- Verify that the `/inventory/items` API operation successfully returns a JSON object with item names and descriptions.



Note: If the call to `http://localhost:9081/inventory/items` fails to return a list of inventory items, you must review the inventory application that you created in the `~/ThinkIBM/inventory` directory. Follow the instructions

in section 3.3.2 to re-publish the inventory application.

Appendix A.6 - Verify that the collective controller and collective member are running

In the previous section, you confirmed that the LoopBack API implementation of the `/inventory/items` operation works correctly.

- Call #3: Consumer app --> API Gateway `/inventory/items` --> IBM HTTP Server --> **Collective plug-in** --> **Collective server** --> Inventory LoopBack app

In this section, you verify whether the collective controller and collective member are working properly.

1. Verify that the collective server is running.

```
cd ~/wlpn/
ls
wlpn-server status inventory-576d5dc5e4b05a78b037514b-146678
5559733-1
```

Note: Replace the collective server name with the one in your `~/wlpn` directory.

Verify that the Collective plug-in in your HTTP server sees your collective server. That is, your server is part of the collective.

2. Open `http://localhost:80/server-status/` Check to see whether there is an entry for the server `inventory-xyz-1`, where xyz is the value of your collective member.

The screenshot shows the Apache Status page for the URL `localhost/server-status`. The page displays a JSON-like structure of server configurations. A specific section for a collective member is highlighted with a red box. Within this box, the 'state' field of the 'servers' object is set to 'STARTED'. Below it, the 'applications' object contains a single entry for 'inventory' with its 'state' also set to 'STARTED'. This indicates that the collective member is running.

```

{
  "clusters": {
    "/cell/defaultCollective/cluster/xubuntu-vm,%2Fhome%2Fstudent%2F.liberty%2Fwlpa%2Fusr:controller": {
      "servers": {
        "/cell/defaultCollective/node/xubuntu-vm,%2Fhome%2Fstudent%2F.liberty%2Fwlpa%2Fusr/server/controller": {
          "state": "STARTED",
          "weight": 2,
          "maintenanceMode": "normal",
          "cloneID": "89443dc8-28be-442f-83e7-5d0062d3949e",
          "averageResponseTimeInMillis": 0,
          "sessionAffinityCookies": "JSESSIONID",
          "outstandingRequests": 0,
          "applications": {}
        }
      }
    },
    "/cell/defaultCollective/cluster/apic.inventory-576d5dc5e4b05a78b037514b-1466785559733": {
      "servers": {
        "/cell/defaultCollective/node/xubuntu-vm,%2Fhome%2Fstudent%2Fwlpa/server/inventory-576d5dc5e4b05a78b037514b-1466785559733-1": {
          "state": "STARTED",
          "weight": 2,
          "maintenanceMode": "normal",
          "averageResponseTimeInMillis": 14,
          "sessionAffinityCookies": "JSESSIONID",
          "outstandingRequests": 0,
          "applications": {
            "inventory": {
              "state": "STARTED",
              "outstandingRequests": 0
            }
          }
        }
      }
    }
  }
}

```

Appendix A.7 - Restart the IBM HTTP server, the collective controller, and collective member

At this point, you completed the following verification steps:

- In *section A.2*, you confirmed that the consumer application completes the call to the OAuth `authorize` operation successfully.
- In *section A.3*, you confirmed that the consumer application sent an *OAuth Authorization Bearer* token in the `/inventory/items` API call.
- In *section A.4*, you verified that the product definition added the correct `app-id` value in the HTTP header in exercise 4.
- In *section A.5*, you confirmed that the LoopBack API implementation of `/inventory/items` works properly.
- In *section A.6*, you verified that the collective member is running.

At this point, restart the HTTP server, the collective controller, and the collective member with this script:

1. Open a terminal window.
2. Run the `restart_inventory` script in the `~/bin` directory.

```
cd ~/bin
./restart_inventory
```

Note: The restart script requires the operating system password. Enter `P`

assw0rd! when prompted.

3. After the collective controller, collective member restarts, run the `consumer_app` in exercise 7 to test the `/inventory/items` API operation.

