

experiment Lab **schedule** 1 hour 30 minutes **universal_currency_alt** No cost
show_chart Introductory

Overview

It is important to identify the caller during an API call.

In this lab, you create an API product and associate it with an API proxy. You will then create an application and associate it with the API product, and use the application's API key when calling the API.

Objectives

In this lab, you learn how to perform the following tasks:

- Create an API product for an API proxy or group of API proxies.
- Create a developer and an application.
- Associate an application with an API product.
- Use a VerifyAPIKey policy to restrict access to registered applications.

Setup

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, **1:15:00**), and make sure you can finish within that time.
There is no pause feature. You can restart if needed, but you have to start at the beginning.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.
If you use other credentials, you'll receive errors or **incur charges**.
7. Accept the terms and skip the recovery resource page.

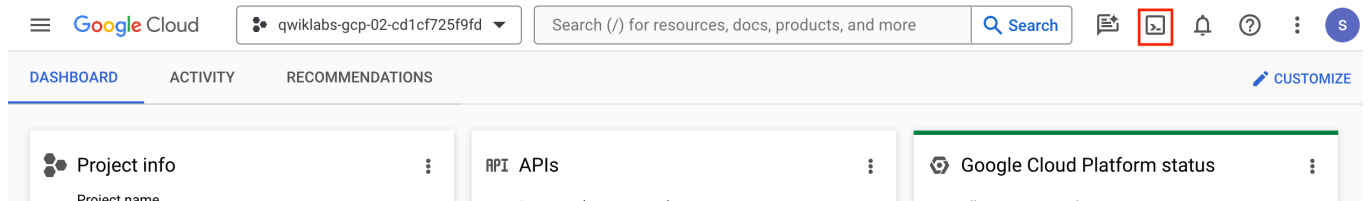
Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



2. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-44776a13dea667a6.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
google1623327_student@cloudshell:~ (qwiklabs-gcp-44776a13dea667a6) $
```

gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

- You can list the active account name with this command:

```
gcloud auth list
```

content_c

Output:

```
Credentialed accounts:
- @.com (active)
```

Example output:

```
Credentialed accounts:
- google1623327_student@qwiklabs.net
```

- You can list the project ID with this command:

content_c

```
gcloud config list project
```

Output:

```
[core]  
project =
```

Example output:

```
[core]  
project = qwiklabs-gcp-44776a13dea667a6
```

Note: Full documentation of **gcloud** is available in the gcloud CLI overview guide .

Preloaded assets

These assets have already been added to the Apigee organization:

- The **retail-v1** API proxy
- The TS-Retail target server in the eval environment (used by retail-v1)

The **highlighted** items in the list are used during this lab.

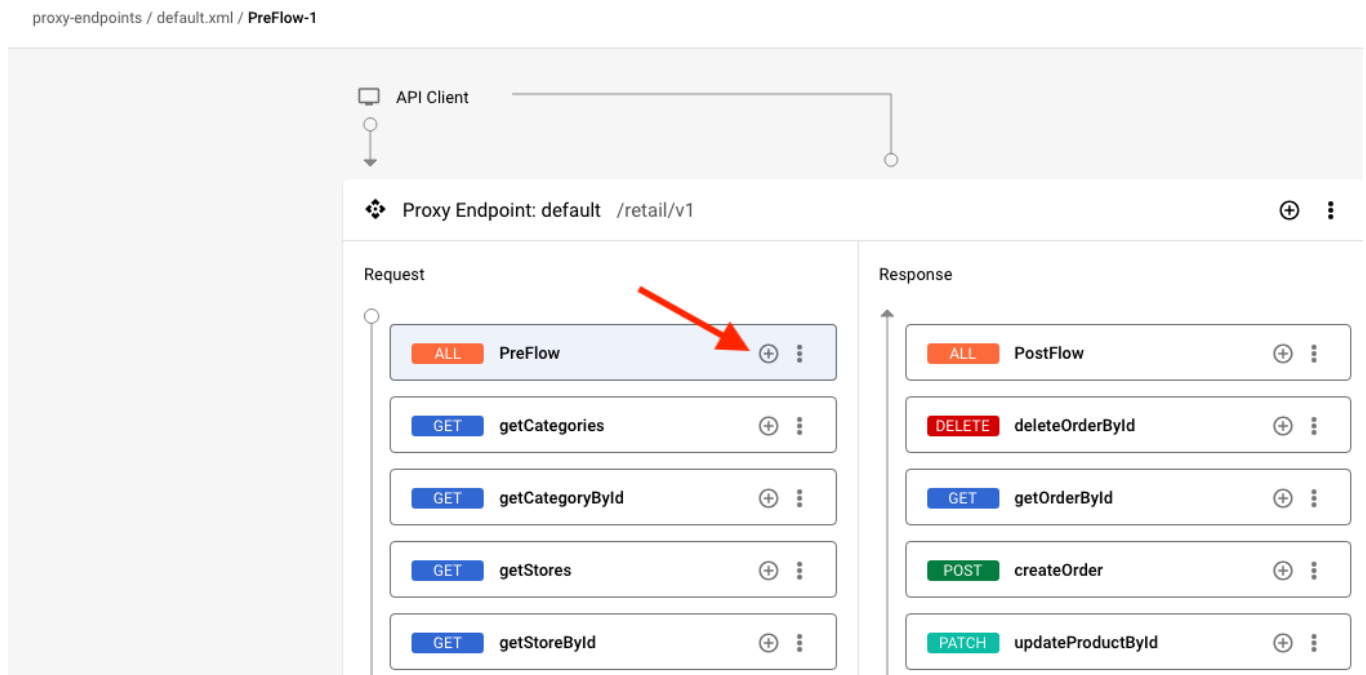
Note: Revision 1 of the retail-v1 proxy is marked as deployed, and is immutable. If you ever make a mistake in your proxy code that you can't recover from, you can select revision 1 and restart editing from there.

Task 1. Add a VerifyAPIKey policy to your API proxy

In this task, you add a VerifyAPIKey policy to force callers to present an API key when accessing the API.

Add the VerifyAPIKey policy

1. In the Google Cloud console, on the **Navigation menu** (≡), select **Integration Services > Apigee > Proxy Development > API proxies**.
2. Select the **retail-v1** proxy.
3. Click the **Develop** tab.
4. In the Navigator pane, click **Proxy endpoints > default > PreFlow**.
5. On the **Request PreFlow**, click **Add Policy Step (+)**.



The VerifyAPIKey policy should be one of the first policies executed, because you want to avoid most of the processing if the caller is not allowed to use the API proxy.

6. In the **Add policy step** pane, select **Create new policy**, and then select **Security > Verify API Key**.

7. Specify the following values:

Property	Value
Name	VA-VerifyKey
Display name	VA-VerifyKey

It is important to correctly set the policy step name. The VerifyAPIKey policy will populate variables that include the step name, and you will use those variables in later labs.

Note: You will use a naming convention for policies in these labs. The prefix, VA, indicates that the policy type is VerifyAPIKey. The rest of the step name, VerifyKey, indicates what the policy is intended to do. This naming convention is useful because the step elements in the endpoint flows don't show the type of policy being attached.

8. Click **Add**.

9. Click on **Policies > VA-VerifyKey** and look at the **APIKey** element. The configuration currently expects the API key in a query parameter named **apikey**.

10. Change the policy configuration so it expects the API key in a **header** named **apikey**. Replace:

```
<APIKey ref="request.queryparam.apikey"/>
```

with:

```
<APIKey ref="request.header.apikey"/>
```

content_c

Note: In real-world proxy development, any API key specified in a query parameter will probably appear in access logs, because the entire URL is generally logged. This can result in API keys being compromised.

It is a better practice to specify the API key in a field that is less likely to be logged, such as a header.

11. To save the updates, click **Save**, and then click **Save as New Revision**.

12. Click **Deploy**.

13. To specify that you want the new revision deployed to the eval environment, select **eval** as the **Environment**, and then click **Deploy**.

14. Click **Confirm**.

You will return to the API proxy later in the lab.

Task 2. Create an API product for your proxy

In this task, you create an API product associated with your API proxy.

API products can be associated with developer applications, providing access to your APIs.

Check runtime status

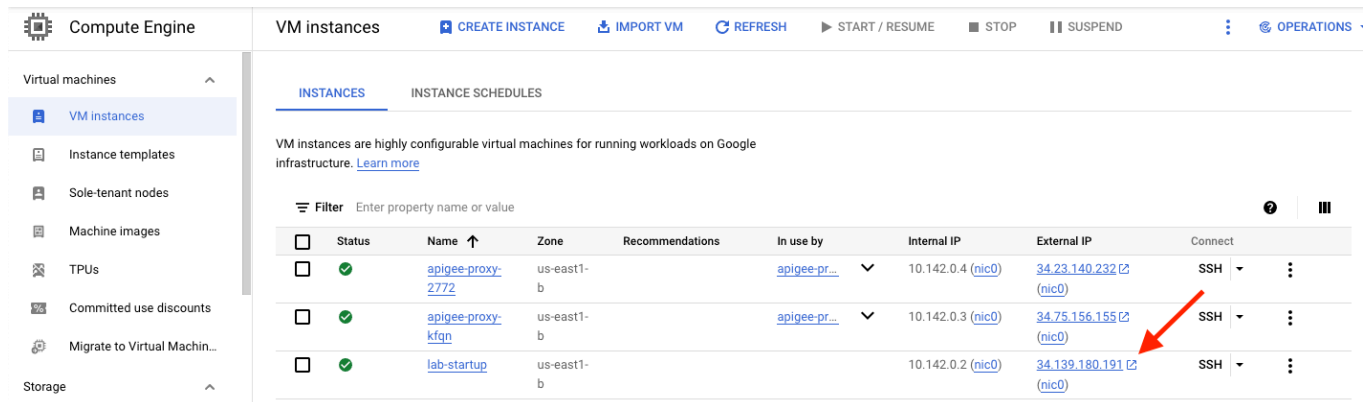
Certain assets, including API products, developers, developer apps, and KVMs, cannot be saved until the runtime is available.

For example, when navigating to the API products page, you might see an error message that reads "Products were not loaded successfully."

This is an error you should see when you are waiting for the runtime instance to be available. Once the runtime is available, refreshing the page will remove the error.

If you get this type of error, you can check the status of provisioning by navigating to the Lab Startup Tasks dashboard at the external IP address of the **lab-startup** VM instance.

1. In the Google Cloud Console navigate to **Compute Engine > VM instances**.
2. To open the Lab Startup Tasks dashboard, click on the **External IP** for the **lab-startup** VM.



3. If you see a redirect notice page, click the link to the external IP address.

A new browser window will open. Lab startup tasks are shown with their progress.

- *Create proxies, shared flows, target servers* should be complete when you first enter the lab, allowing you to use the Apigee console for tasks like proxy editing.
- *Create API products, developers, apps, KVMs, KVM data* indicates when the runtime is available and those assets may be saved.
- *Proxies handle API traffic* indicates when the eval environment has been attached to the runtime and the deployed proxies can take runtime traffic.

- Lab Startup Tasks -			
Progress	Time	State	Task
<div></div>	05:02	completed	Create proxies, shared flows, target servers (environment available)
<div></div>	30:49	completed	Create API products, developers, apps, KVMs, KVM data (runtime is available)
<div></div>	31:11	started	Proxies handle API traffic (environment attached to runtime)
<div></div>	03:34	completed	Provide access to lab
<div></div>	30:05	started	Full provisioning of Apigee org qwiklabs-gcp-02-d23d90c73c5a in us-west4
<div></div>	01:41	completed	Create Apigee load balancer at api-test-qwiklabs-gcp-02-d23d90c73c5a.apigee-api
<div></div>	00:14	completed	Connect load balancer to runtime instance

**Before continuing, you need to wait for Create API products, developers, apps, KVMs, KVM data to complete.*

While you are waiting

- What is an API product?

- Managing API products
- Controlling access to your APIs by registering apps

Create the API product

1. In the Google Cloud console, on the **Navigation menu** (≡), select **Integration Services > Apigee > Distribution > API products**.
2. To create a new API product, click **+Create**.
3. In the **Product details** section, specify the following:

Property	Value
Name	retail-fullaccess
Display Name	retail (full access)
Description	full access to retail API
Environment	<i>select eval, and then click OK</i>
Access	<i>select Public</i>

Leave **Automatically approve access requests** selected.

4. In the **Operations** section, click **+Add an Operation**.

Operations are used to specify which requests in which API proxies are allowed for an application associated with the API product.

Note: Confirm that the button is in the "Operations" section, not the "GraphQL Operations" section.

5. In the **Operation** pane, specify the following values:

Property	Value
API Proxy	<i>select the retail-v1 API proxy</i>

Path	<i>/**</i>
Methods	<i>select GET, PATCH, POST, PUT, and DELETE</i>

The path expression `"/**"` indicates that any path suffix of any depth is a match for the operation.

In a production environment, you might choose to add each operation that is allowed separately, rather than using this wildcard path expression.

6. Click **Save** to save the operation.

7. In the **Custom Attributes** section for the API product, click **+Add Custom Attribute**.

Custom attributes can be used to attach any data that you would like to be available in the proxy to control access.

In this case, because this is the full access API product for your retail API, you'll create a custom attribute that indicates that the application calling the API should be allowed full access.

8. Specify the following:

Property	Value
Name 1	full-access
Value 1	yes

9. Click **OK** to save the custom attribute.

10. To save the API product, click **Save**.

11. Return to the **API Products** page. Your API product will be listed.

Task 3. Create a second API product for your proxy

In this task, you create a read-only API product associated with your API proxy.

1. To create a new API product, click **+Create**.
2. In the **Product details** pane, specify the following:

Property	Value
Name	retail-readonly
Display Name	retail (read-only)
Description	read-only access to retail API
Environment	<i>select eval</i> , and then click OK
Access	<i>select Public</i>

Leave **Automatically approve access requests** selected.

3. In the **Operations** section, click **+Add an Operation**.
4. In the **Operation** pane, specify the following values:

Property	Value
API Proxy	<i>select the retail-v1 API proxy</i>
Path	<i>/**</i>
Methods	<i>select GET</i>

By specifying only the **GET** method, you are limiting API requests to read-only operations.

5. Click **Save** to save the operation.
6. To save the API product, click **Save**.
7. Return to the **API Products** page. Your second API product will be listed.

Task 4. Create an app developer

In this task, you create an app developer, which will allow you to register an app.

Note: App developers are typically created using a developer portal, and you will be creating your developer portal in a later lab. For now, use the Apigee console to create the developer.

1. On the Navigator menu, click **Distribution > Developers**.
2. To create a new app developer, click **+Create**.
3. Specify the following:

Property	Value
First Name	Joe
Last Name	Developer
Email	joe@example.com
Username	joe

4. To create the app developer, click **Add**.

Task 5. Create an app

In this task, you create an app for your app developer.

1. On the Navigator menu, click **Distribution > Apps**.
2. To create a new app, click **+Create**.

3. In the **App Details** pane, specify the following:

Property	Value
App Name	retail-app
Display Name	Joe's retail app
Developer	<i>select Joe Developer</i>

4. In the **Credentials** section, click **+Add Credential**.


5. In the **Add Credential** pane, for **Product 1**, select **retail (read-only)**.

6. For **Status 1**, select **Approved**.

7. To add the credential, click **Add**.

8. To create the app, click **Create**.

The **Key** and **Secret** are now configured for the app.

9. For the **Key**, click on the not shown icon ()

The API key for Joe's retail app is shown.

10. To copy the key into the clipboard, for the **Key**, click **Copy to clipboard** ()

11. In Cloud Shell, run the following command:

```
export API_KEY={key}
```

content_c

Replace **{key}** with the API key for Joe's app. The command should look like this:

```
export API_KEY=ZyncwYSmswgJVKNCw204dmjPp0XnGihgu5NoWXHXHpUpSd1jL
```

Task 6. Verify the behavior of the proxy

In this task, you will verify that requests made to the proxy provide the correct access.

Start the debug session

1. On the left navigation menu, click **Proxy development > API proxies**.
2. Click on the **retail-v1** proxy.

You should see that the proxy is deployed to the **eval** environment.

3. Click the **Debug** tab, and then click **Start Debug Session**.
4. In the **Start debug session** pane, on the Environment dropdown, select **eval**.

The deployed revision number will also show in the dropdown.

5. Click **Start**.

Test the API

1. Back in Cloud Shell, execute this curl command:

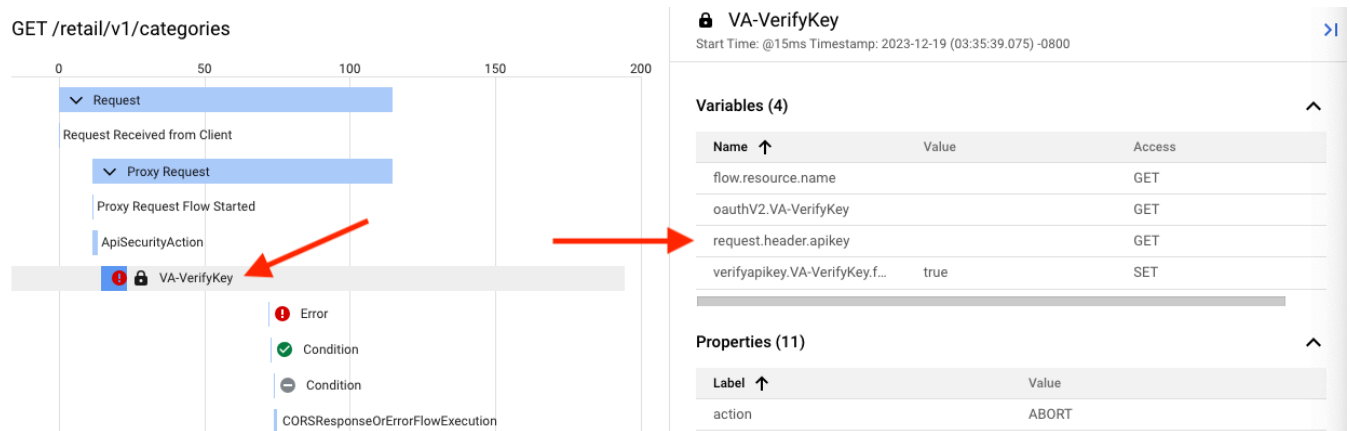
```
curl -i -X GET "https://api-test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

content_c

The API request should return 401.

2. In **Debug**, click on the GET request, and then click **VA-VerifyKey**.

You should see that the policy retrieved the request.header.apikey variable, but the value was empty.



The error message is "Failed to resolve API Key variable request.header.apikey". The API key was not found.

3. Add the apikey header with an incorrect API key by using this command:

```
curl -i -X GET -H "apikey:123" "https://api-test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

content_c

The API request still returns 401, but the error is "Invalid ApiKey." Looking at the Debug tool, the key was found in **request.header.apikey** with the value **123**.

This error means you are passing the API key in the correct location.

Note: If you are not getting the "Invalid ApiKey" error, then the VerifyAPIKey policy is not looking in the right location. Make sure the debug tool shows that the variable request.header.apikey is being read. If it is not, return to the proxy to fix it, save, and redeploy.

4. Verify that the API key has been copied to the environment variable using this command:

```
echo $API_KEY
```

content_c

If the variable is empty, return to Task 5 for the instructions to create the variable.

5. Add the apikey header with the correct API key by using this command:

```
curl -i -X GET -H "apikey:${API_KEY}" "https://api-test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

content_c

With the correct API key, you should see the list of categories.

If you look at the debug tool, you will notice that many extra variables have been populated, including the API product name, app name, and developer email. These variables are all available for use in your proxy.

6. Try a non-GET command with the same API key:

```
curl -i -X DELETE -H "apikey:${API_KEY}" "https://api-test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

content_c

There isn't a *DELETE* /*categories* operation in the backend API, but the request never reached the backend because Joe's app is using the read-only API product, where the only method allowed is *GET*. Joe's request was blocked by the VerifyAPIKey policy, returning the error "*Invalid ApiKey for given resource.*"

Congratulations!

In this lab, you created two API products (one read-only and one with full access), a developer, and an app, and you used a VerifyAPIKey policy in your API proxy to require that only registered apps can call your API.

End your lab