Apigee Lab 4: Using OAuth

experiment Lab schedule 1 hour 30 minutes universal_currency_alt No cost

show_chart Introductory

Overview

In this lab, you use the OAuthV2 policy to allow apps to access the retail API proxy by providing an OAuth token.

Objectives

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

- Protect your API proxy by requiring an OAuth token.
- Invoke an OAuth proxy to retrieve a token.
- Attach an OAuth token to an API request.

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.

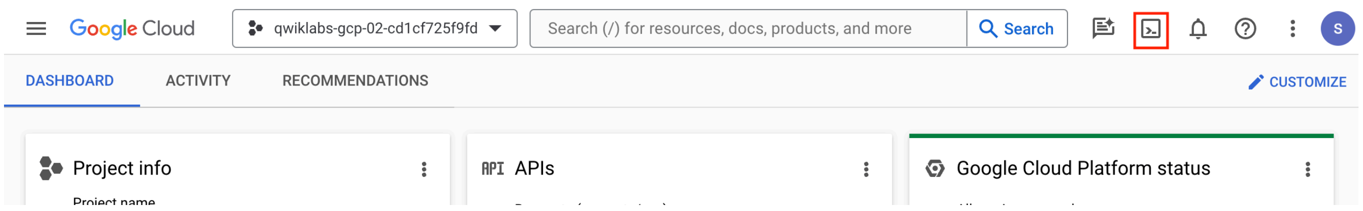
Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

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.

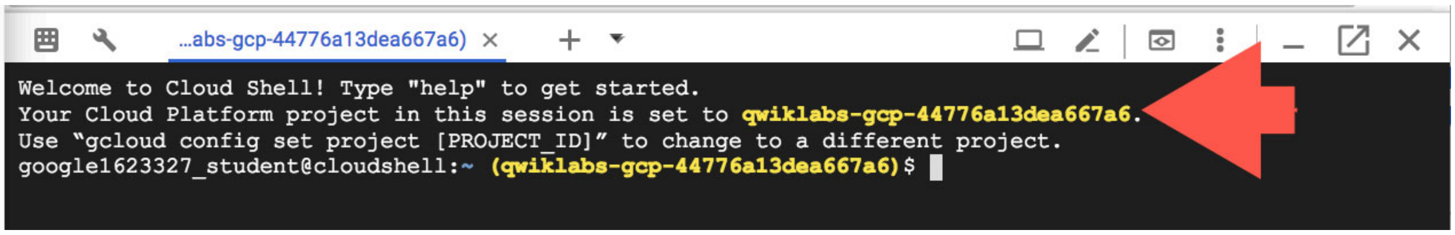
Google Cloud Shell provides command-line access to your Google Cloud resources.

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


Google Cloud
qwiklabs-gcp-02-cd1cf725f9fd
Search (/) for resources, docs, products, and more
Search
S
DASHBOARD
ACTIVITY
RECOMMENDATIONS
CUSTOMIZE
Project info
Project name
API APIs
Google Cloud Platform status

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:

```
gcloud config list project
```

content_c

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
- An **oauth-v1** API proxy (used to generate OAuth tokens for this lab)
- The TS-Retail target server in the eval environment (used by retail-v1)

These assets will be added to the Apigee organization as soon as the runtime is available:

- The API products, developer, and **developer app** (used by retail-v1)

The **highlighted** items 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 an OAuth policy to your API proxy to verify tokens

In this task, to force callers to present an OAuth token when accessing the API, you add an OAuthV2 policy.

Add the OAuthV2 policy in place of 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.

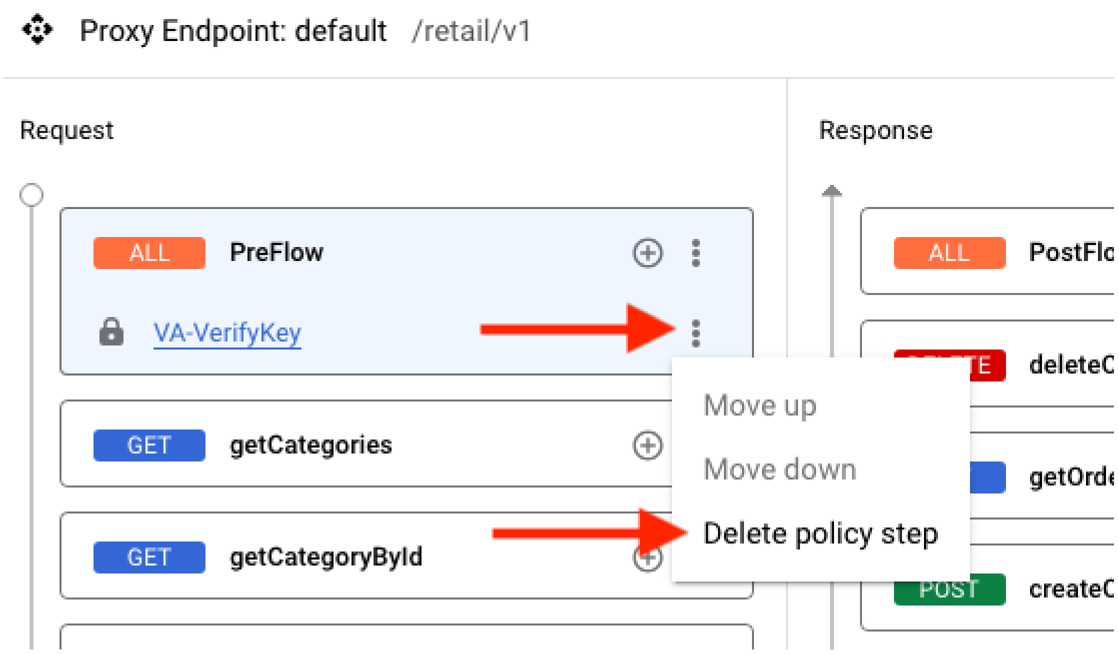
3. Click the **Develop** tab.

   You are modifying the version of the retail-v1 proxy that was created during Labs 1 through 3.

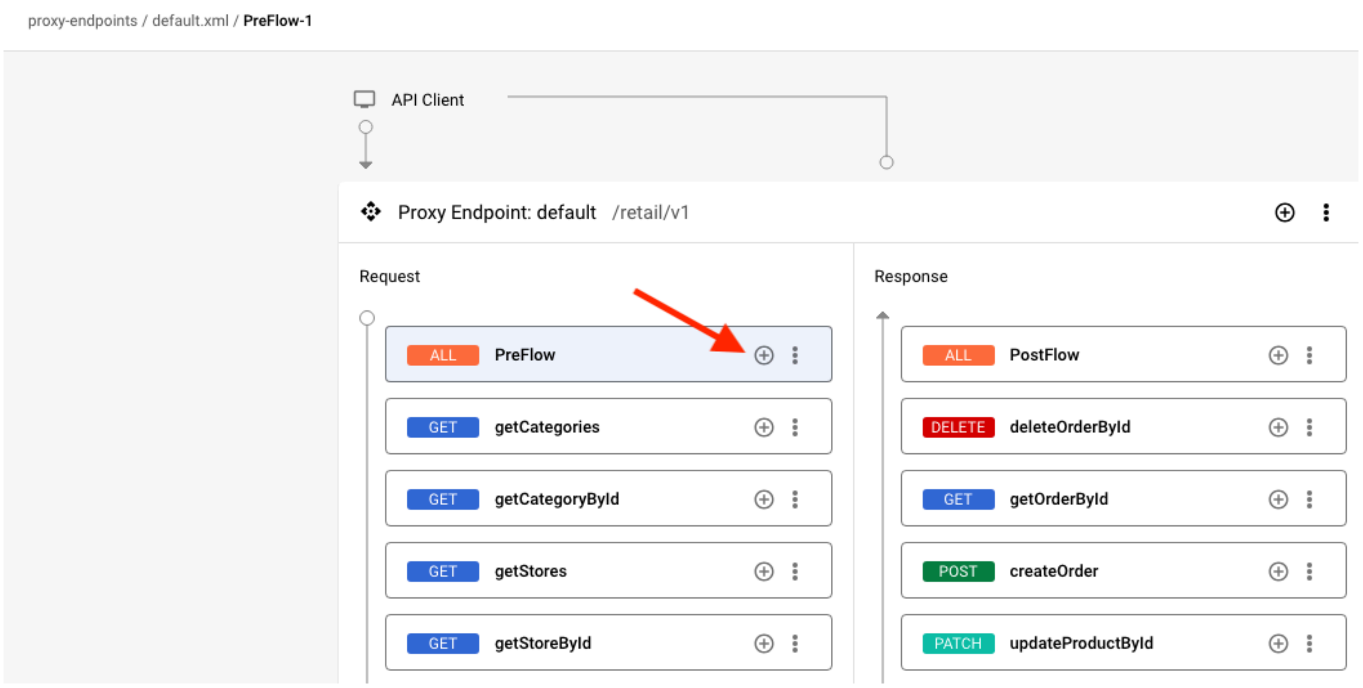
4. In the Navigator pane, click **Proxy endpoints > default > PreFlow**.

   The step in that flow, the **VerifyAPIKey** policy named **VA-VerifyKey**, is graphically represented.

5. To detach the **VA-VerifyKey** policy, click **Policy step actions** (⋮), and then click **Delete policy step**.



6. On the **Request PreFlow**, click **Add Policy Step (+)**.



7. In the **Add policy step** pane, select **Create new policy**, and then select **Security > OAuth v2.0**.

8. Specify the following values:

Property	**Value**
Name	**OAuthV2-VerifyToken**
Display name	**OAuthV2-VerifyToken**

9. Click **Add**.

10. Click on **Policies > OAuthV2-VerifyToken**.

The policy's default Operation is **VerifyAccessToken**, which is the correct operation. By default, the policy will look for the OAuth token in the standard location, which is the **Authorization** header. The value of the header must be:

```
Bearer {token}
```

11. Even though the default configuration of the policy will work, there are elements that are not used with the **VerifyAccessToken** operation. Make the configuration cleaner by replacing the configuration with:

```
<OAuthV2 continueOnError="false" enabled="true" name="OAuthV2-
VerifyToken">
  <Operation>VerifyAccessToken</Operation>
</OAuthV2>
```

content_c

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

13. Click **Deploy**.

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

15. Click **Confirm**.

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

Task 2. Examine the OAuth proxy

In this task, you will examine the OAuth proxy that has been added to the organization.

An OAuth proxy named **oauth-v1** has been added to your Apigee organization. It can be used to create OAuth access tokens for the client credentials grant type, which we will be using in this lab.

Note: This proxy will only create tokens for the client credentials grant type.

1. Navigate to **Proxy development > API Proxies**.

2. Select the **oauth-v1** proxy, and then click the **Develop** tab.

3. Click on the **POST /token cc grant** flow.

   The flow will be executed if:

   - request.verb is *POST*
   - The proxy.pathsuffix is */token*
   - The payload has a form parameter with name *grant_type* and value *client_credentials*

   These follow the OAuth 2.0 specification.

4. Click on the **OAuthV2 policy**.

   Looking at the policy configuration, the OAuthV2 policy is using the **GenerateAccessToken** operation. When this policy runs, it will generate an access token for the **client_credentials** grant type. The token will expire in 60 minutes.

   Note: In a production environment, it would be more appropriate to choose a shorter expiration time.

Task 3. Generate a new OAuth token

In this task, you use the OAuth proxy to create an OAuth token for your application.

Wait for the developer app to be available

Check runtime status

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

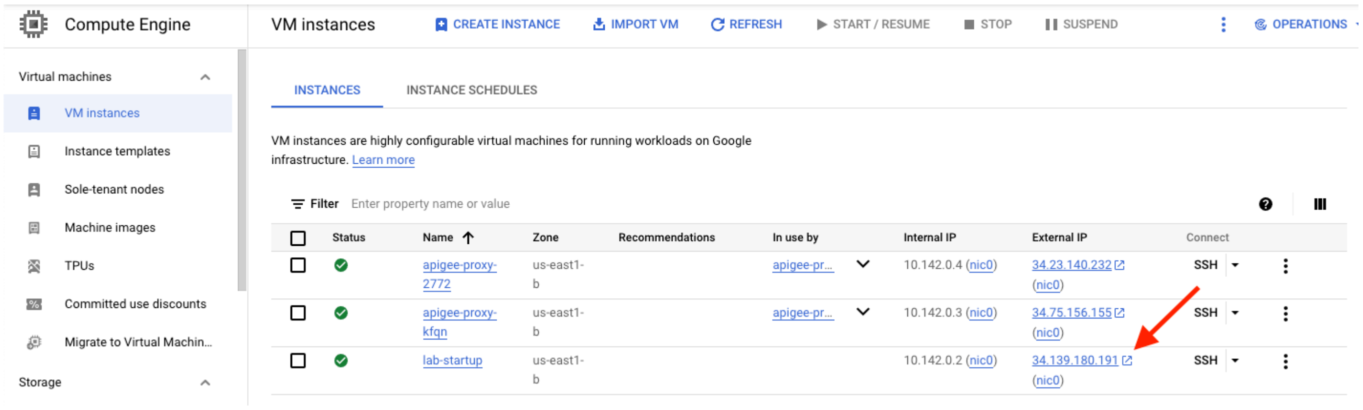
For example, when navigating to the API products page, you might see an error message that reads "Products were not loaded successfully."

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.



VM instances are highly configurable virtual machines for running workloads on Google infrastructure. Learn more

	Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
☐	✓	apigee-proxy-2772	us-east1-b		apigee-pr...	10.142.0.4 (nic0)	34.23.140.232 (nic0)	SSH
☐	✓	apigee-proxy-kfqn	us-east1-b		apigee-pr...	10.142.0.3 (nic0)	34.75.156.155 (nic0)	SSH
☐	✓	lab-startup	us-east1-b			10.142.0.2 (nic0)	34.139.180.191 (nic0)	SSH

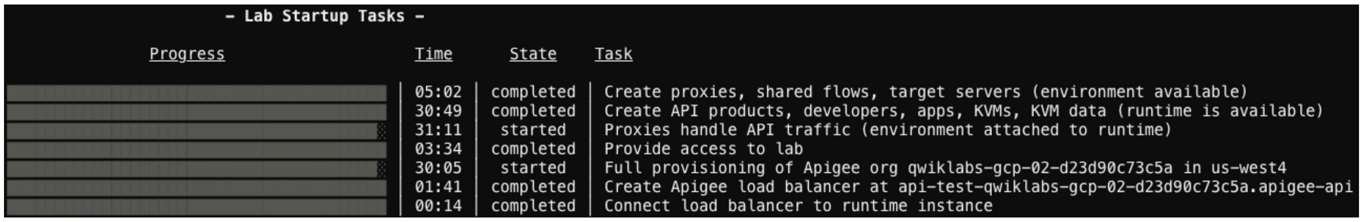
3. If you see a redirect notice page, click the link to the external IP address.

   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

                                     | 05:02 | completed | Create proxies, shared flows, target servers (environment available)
                                     | 30:49 | completed | Create API products, developers, apps, KVMs, KVM data (runtime is available)
                                     | 31:11 |  started  | Proxies handle API traffic (environment attached to runtime)
                                     | 03:34 | completed | Provide access to lab
                                     | 30:05 |  started  | Full provisioning of Apigee org qwiklabs-gcp-02-d23d90c73c5a in us-west4
                                     | 01:41 | completed | Create Apigee load balancer at api-test-qwiklabs-gcp-02-d23d90c73c5a.apigee-api
                                     | 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

- Read the OAuthV2 policy reference.
- Watch the YouTube video Apigee Edge - 4MV4D - API Security - OAuth 2.0 - Explained in 4 Minutes for Beginners - S24E02.
- Read The OAuth 2.0 Authorization Framework.

Store the app's key and secret in shell variables

1. Navigate to **Distribution** > **Apps**.

2. Select **Joe's retail app**.

   This app is associated with the read-only retail product.

3. To copy the Key into the clipboard, for the **Key**, click **Copy to clipboard** (⧉).

4. In **Cloud Shell**, create a shell variable that contains the key's value:

```
export APP_KEY=REPLACE
```

content_c

Replace the word *REPLACE* with the key you copied. Verify that the key matches the key in the Apigee console, and that there are no leading or trailing spaces.

5. To copy the Secret into the clipboard, for the **Secret**, click **Copy to clipboard** ( ).

6. In **Cloud Shell**, create a shell variable that contains the secret's value:

```
export APP_SECRET=REPLACE
```

content_c

Replace the word *REPLACE* with the secret you copied. Verify that the secret matches the secret in the Apigee console, and that there are no leading or trailing spaces.

7. Print the saved variables to confirm that they are correct:

```
echo "(${APP_KEY}:${APP_SECRET})"
```

content_c

You should see the app's key and secret inside the parentheses and separated by a colon, with no spaces.

OAuth requires that the application's key and secret be passed in a **Basic Authentication header**. A Basic Auth header uses the header named **Authorization**, and the value is the string "Basic " followed by a Base64-encoded string. The string before Base64 encoding is the key and secret, separated by a colon (":").

8. You can use a command-line tool to Base64-encode the username and password string:

```
echo -n "${APP_KEY}:${APP_SECRET}" | base64
```

content_c

The **base64** command encodes the string and prints it, wrapping the text at 76 columns.

9. To save the *key* and *secret* into the *.bashrc* file, run this command:

```
echo "export APP_KEY=${APP_KEY}" >> .bashrc
echo "export APP_SECRET=${APP_SECRET}" >> .bashrc
```

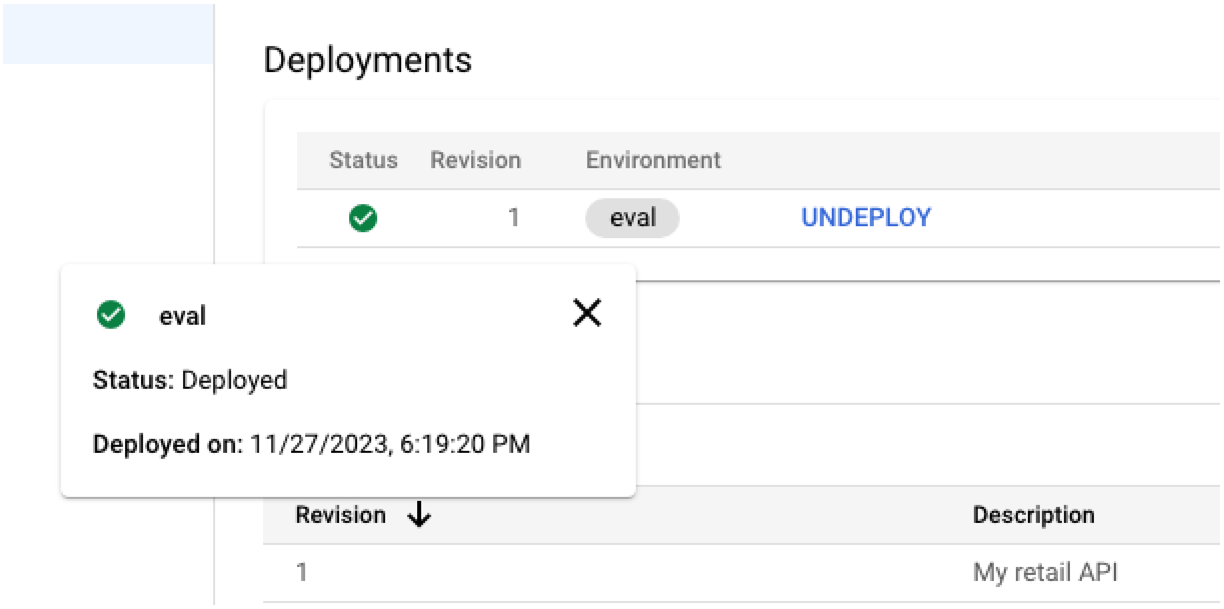
content_c

If *Cloud Shell* is closed or disconnects, the *APP_KEY* and *APP_SECRET* variables will be automatically repopulated.

Next, you will confirm that the *oauth-v1* and *retail-v1* are deployed.

Check deployment status

A proxy that is deployed and ready to take traffic will show a green status.



Deployments

Status	Revision	Environment	
✓	1	eval	UNDEPLOY

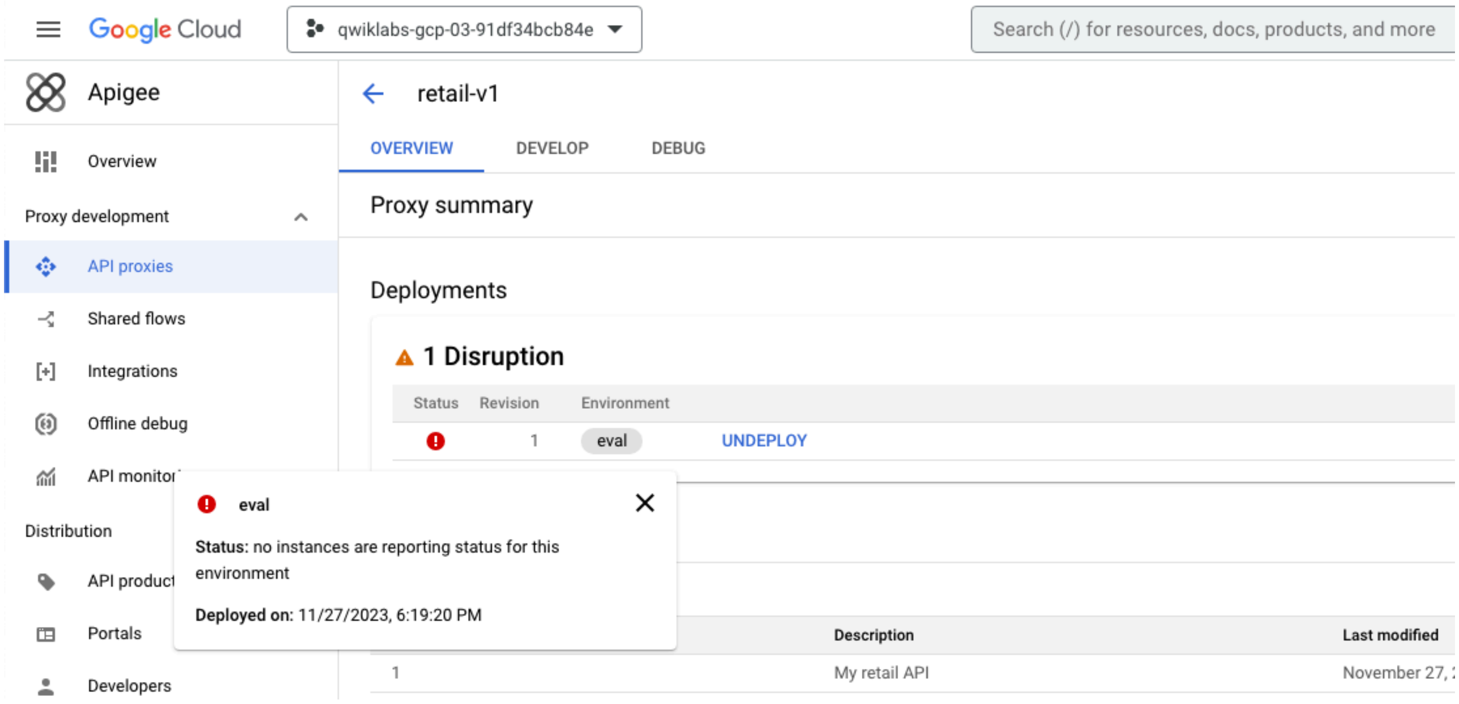
eval

Status: Deployed

Deployed on: 11/27/2023, 6:19:20 PM

Revision	Description
1	My retail API

When a proxy is marked as deployed but the runtime is not yet available and the environment is not yet attached, you will see a yellow caution sign. Hold the pointer over the **Details** text to see the current status.



Proxy summary

Deployments

1 Disruption

Status	Revision	Environment	
!	1	eval	UNDEPLOY

eval

Status: no instances are reporting status for this environment

Deployed on: 11/27/2023, 6:19:20 PM

	Description	Last modified
1	My retail API	November 27,

If the proxy is deployed and shows as green, your proxy is ready for API traffic.

If your proxy is not deployed because there are no runtime pods, you can check the status of provisioning by following these instructions.

You can check the status of provisioning by checking the Lab Startup Tasks dashboard.

When *Proxies handle API traffic* is complete, your proxies should be ready to take traffic.

Get a token

1. Execute this curl command:

```
curl -u "${APP_KEY}:${APP_SECRET}" -H 'Content-Type:
application/x-www-form-urlencoded' -X POST "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/oauth/v1/token" -d
"grant_type=client_credentials" | json_pp
```

content_c

The *-u* parameter automatically creates a Basic Auth header using the app's key and secret, and adds it to the request.

The request is **POST /token**, with the request body being a form parameter indicating the grant type. The Content-Type, **application/x-www-form-urlencoded**, indicates the content type of the request.

The fields should resemble this:

```
{
    "access_token" : "lC8rxmTm53HAj6Qphfes3Q4ALemt",
    "token_type" : "Bearer",
    "expires_in" : "3599",
    "scope" : ""
}
```

Note: Curl does not pretty print the JSON response. To pretty print the JSON payload in the response, you can remove the -i parameter (which shows headers in the response) and then pipe the curl command to json_pp, a JSON pretty printer.

The **access_token** may be used to call the retail API. It will provide the permissions that are granted by the API product(s) associated with the application.

2. To automatically copy a token into a shell variable, use this command:

```
export TOKEN=$(curl -q -s -u "${APP_KEY}:${APP_SECRET}" -H
'Content-Type: application/x-www-form-urlencoded' -X POST
"https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/oauth/v1/token" -d
"grant_type=client_credentials" | jq --raw-output
".access_token"); echo "TOKEN=$TOKEN"
```

content_c

The *jq* command extracts the **access_token** value from the JSON response.

Task 4. Test the retail API

In this task, you will test the retail API, confirming that a valid OAuth token is now required to call the API.

1. Call the API without a token using this command:

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

content_c

This command returns a 401 error:

```
{"fault":{"faultstring":"Invalid access token","detail":{"errorcode":"oauth
```

2. Call the API with a bad token using this command:

```
curl -i -H "Authorization: Bearer 12345" -X GET "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

content_co

This command also returns a 401 error:

```
{"fault":{"faultstring":"Invalid Access Token","detail":{"errorcode":"keyma
```

3. Call the API with the valid token using this command:

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

content_co

This command also returns a 401 error, but with a different format:

```
{"error":"invalid_credentials","error_description":"Credentials missing
or incorrect."}
```

If you have correctly attached a valid token, you receive this error. Why is an error still happening, even though the token was attached correctly?

Whenever you are not sure what is happening in the API proxy, the debug tool may be used to trace an API call.

4. Click **Proxy development > API Proxies**.

5. Select the **retail-v1** proxy, then click the **Debug** tab, and then click **Start Debug Session**.

6. In the **Start debug session** pane, on the Environment dropdown, select **eval**.

   The deployed revision number will also show in the dropdown.

7. Click **Start**.

   A debug session will last a maximum of 10 minutes or until 15 transactions are received, whichever comes first.

   Debug will trace requests being sent to the API proxy.

8. Back in Cloud Shell, repeat the curl command:

```
export TOKEN=$(curl -q -s -u "${APP_KEY}:${APP_SECRET}" -H
'Content-Type: application/x-www-form-urlencoded' -X POST
"https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/oauth/v1/token" -d
"grant_type=client_credentials" | jq --raw-output
".access_token"); echo "TOKEN=$TOKEN"
curl -i -H "Authorization: Bearer ${TOKEN}" -X GET "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

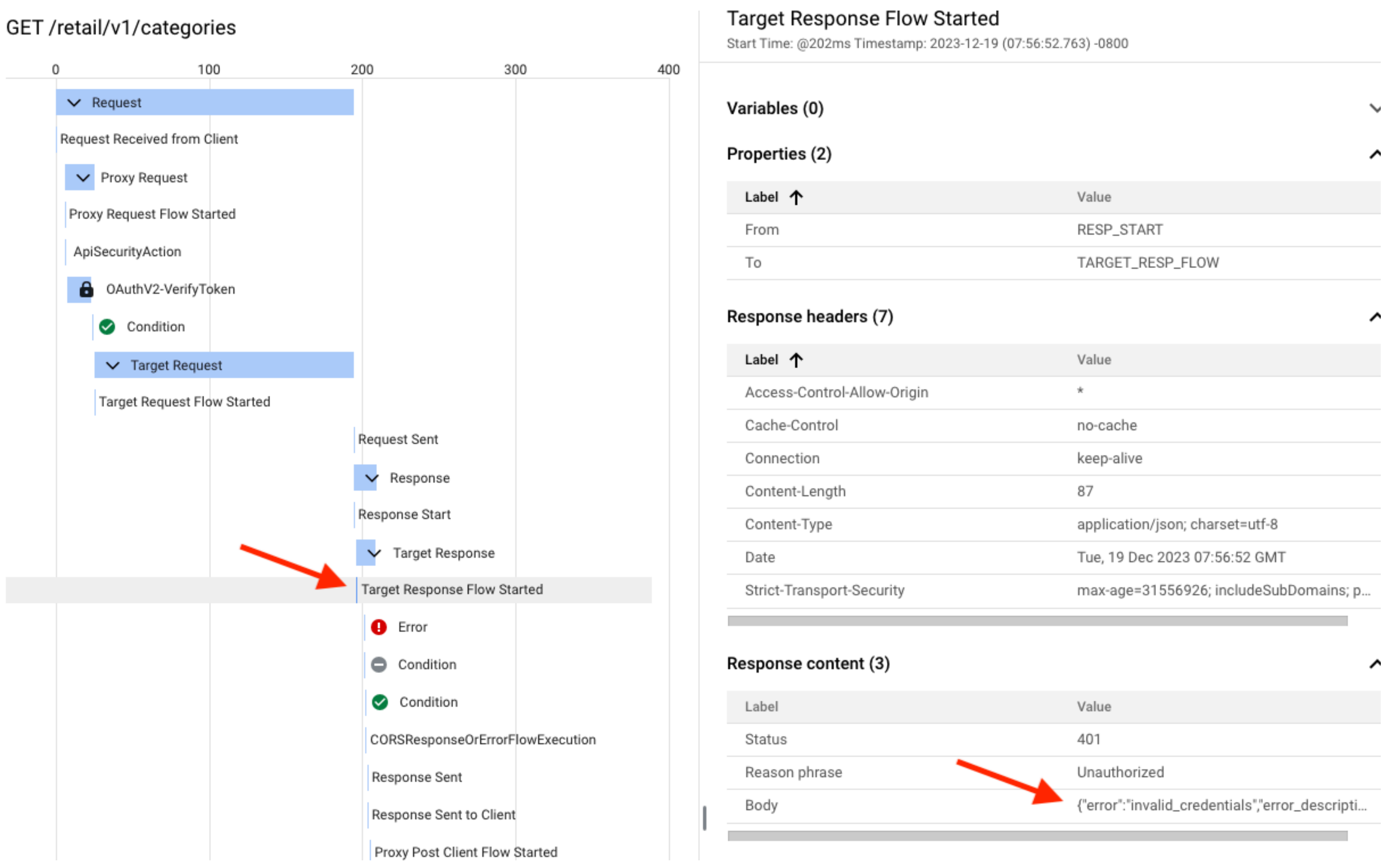
content_c

   Use the debug tool to try to investigate what is happening. Try to determine what is happening before moving to the next task.

Task 5. Understand the error

In this task, you determine the source of the error.

Did you understand what happened? The transaction looks like this:



GET /retail/v1/categories

Target Response Flow Started

Start Time: @202ms Timestamp: 2023-12-19 (07:56:52.763) -0800

Variables (0)

Properties (2)

Label	Value
From	RESP_START
To	TARGET_RESP_FLOW

Response headers (7)

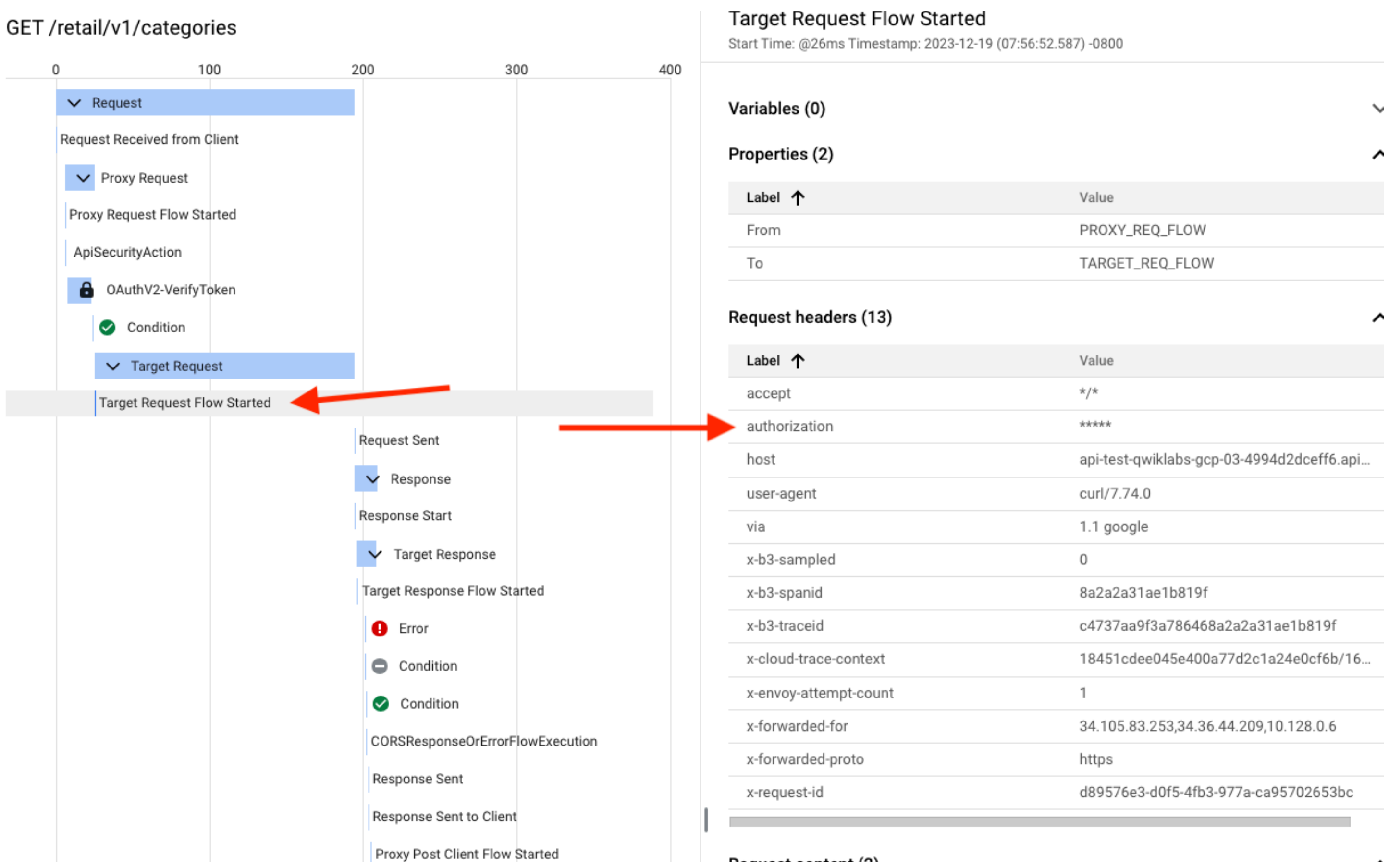
Label	Value
Access-Control-Allow-Origin	*
Cache-Control	no-cache
Connection	keep-alive
Content-Length	87
Content-Type	application/json; charset=utf-8
Date	Tue, 19 Dec 2023 07:56:52 GMT
Strict-Transport-Security	max-age=31556926; includeSubDomains; p...

Response content (3)

Label	Value
Status	401
Reason phrase	Unauthorized
Body	{"error":"invalid_credentials","error_descripti...

Note: The OAuthV2 step does not have a red circle with an exclamation point. The policy successfully verified the token and did not raise a fault. The factory icon indicates that the backend target was called, and the 401 response came from the target.

The error message indicates **invalid credentials**. What credentials did you send to the backend?


GET /retail/v1/categories

Request
Request Received from Client
Proxy Request
Proxy Request Flow Started
ApiSecurityAction
OAuthV2-VerifyToken
Condition
Target Request
Target Request Flow Started
Request Sent
Response
Response Start
Target Response
Target Response Flow Started
Error
Condition
Condition
CORSResponseOrErrorFlowExecution
Response Sent
Response Sent to Client
Proxy Post Client Flow Started

Target Request Flow Started
Start Time: @26ms Timestamp: 2023-12-19 (07:56:52.587) -0800

Variables (0)

Properties (2)

Label	Value
From	PROXY_REQ_FLOW
To	TARGET_REQ_FLOW

Request headers (13)

Label	Value
accept	*/*
authorization	*****
host	api-test-qwiklabs-gcp-03-4994d2dceff6.api...
user-agent	curl/7.74.0
via	1.1 google
x-b3-sampled	0
x-b3-spanid	8a2a2a31ae1b819f
x-b3-traceid	c4737aa9f3a786468a2a2a31ae1b819f
x-cloud-trace-context	18451cdee045e400a77d2c1a24e0cf6b/16...
x-envoy-attempt-count	1
x-forwarded-for	34.105.83.253,34.36.44.209,10.128.0.6
x-forwarded-proto	https
x-request-id	d89576e3-d0f5-4fb3-977a-ca95702653bc

Explanation

The Authorization header was sent to the backend target in the request! By default, any headers, query parameters, and payload that are sent in a request are passed through to the target.

This API proxy requires that a caller send an OAuth token in an Authorization header to gain access to the API. As you will see in a future lab, the backend service uses the Authorization header for Basic Authentication. Even if it was using the Authorization header for OAuth, it would have no way to verify the Apigee token.

A backend service typically doesn't know details about the security added in API proxy that calls it. The backend service detected an invalid Authorization header, and therefore returned an error.

It is always a good policy to strip off any credentials that are used at the API proxy level before sending the request to the backend. In many cases, the backend might ignore the data. However, this sensitive data could be logged somewhere, or be seen by the operations team. It is recommended that you remove sensitive data from the request message after it is used and before you forward it to any external service.

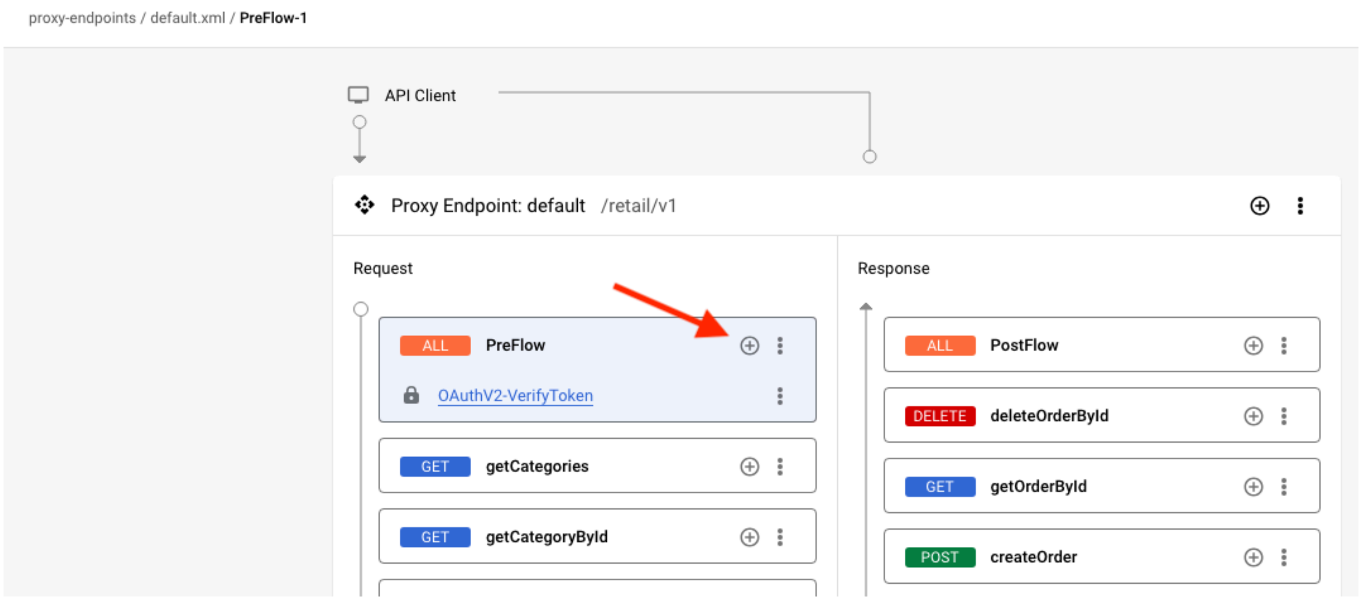
You might have determined that something unusual was happening because the error format was different from the other errors you saw, but the debug tool is typically the best way to debug your proxies.

Task 6. Remove the Authorization header

In this task, you remove the Authorization header and API key before sending the request to the target.

Note: It is a best practice to remove the Authorization header, API key, or any other information not relevant to the backend target before sending the request to the target.

1. Click the **Develop** tab.
2. In the Navigator pane, click **Proxy endpoints > default > PreFlow**.
3. On the **Request PreFlow**, click **Add Policy Step (+)**.



The policy currently in that flow, the **OAuthV2** policy named **OAuthV2-VerifyToken**, is graphically represented.

4. In the **Add policy step** pane, select **Create new policy**, and then select **Mediation > Assign Message**.

5. Specify the following values:

Property	Value
Name	**AM-RemoveAuth**
Display name	**AM-RemoveAuth**

6. Click **Add**.

7. Click **Policies > AM-RemoveAuth**.

8. Replace the default configuration with this:

   content_c

   ```
   <AssignMessage continueOnError="false" enabled="true" name="AM-
   RemoveAuth">
     <Remove>
       <Headers>
         <Header name="Authorization"/>
         <Header name="apikey"/>
       </Headers>
     </Remove>
     <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
     <AssignTo createNew="false" transport="http" type="request"/>
   </AssignMessage>
   ```

   Any Authorization or apikey header will be removed when this policy executes.

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

10. Click **Deploy**.

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

12. Click **Confirm**.

13. Click the **Overview** tab.

14. After waiting for the change to be deployed, make the request again:

```
export TOKEN=$(curl -q -s -u "${APP_KEY}:${APP_SECRET}" -H
'Content-Type: application/x-www-form-urlencoded' -X POST
"https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/oauth/v1/token" -d
"grant_type=client_credentials" | jq --raw-output
".access_token")
curl -i -H "Authorization: Bearer ${TOKEN}" -X GET "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories"
```

content_c

You should now see the categories in a **200 OK** response.

Congratulations!

In this lab, you learned how to use an OAuth proxy to get an access token for your app, protect your API using the OAuthV2 VerifyAccessToken operation, and how to send a token with your API.

End your lab