Overview

In this lab, you create an integrated developer portal, publish an API product and OpenAPI specification to the portal, and make API calls using the live documentation in the portal.

Objectives

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

- Support CORS in an API proxy.
- Create an integrated developer portal.
- Publish APIs as API products on the integrated developer portal.
- Sign in as an app developer and use live documentation in the developer portal.

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.

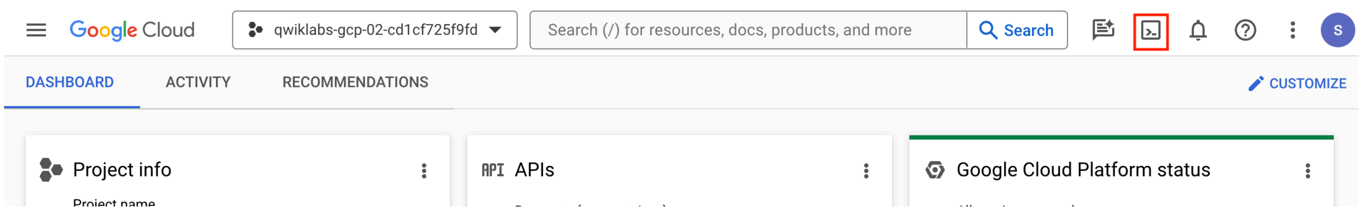
Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

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.

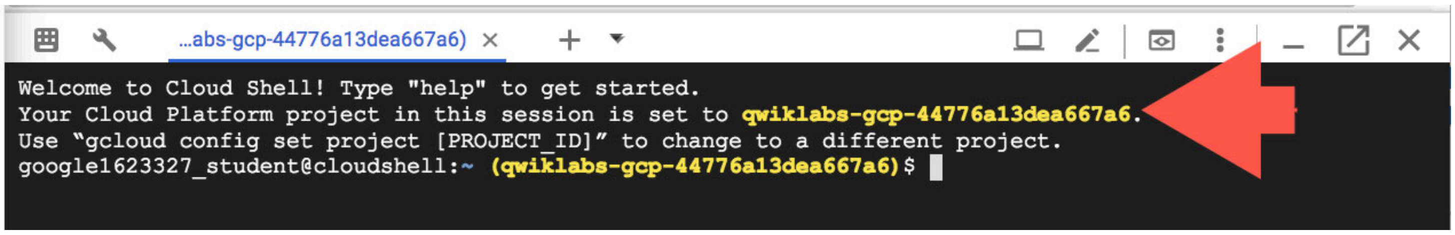
Google Cloud Shell provides command-line access to your Google Cloud resources.

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
API APIs
Google Cloud Platform status
Project name

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
- The oauth-v1 API proxy (for generating OAuth tokens)
- The backend-credentials shared flow (used by retail-v1)
- 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 ProductsKVM key value map in the eval environment (used by backend-credentials)
- The ProductsKVM key value map entries backendId and backendSecret

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 CORS to proxy

In this task, you add CORS (cross-origin resource sharing) to the retail-v1 proxy.

CORS is a protocol that uses HTTP headers to indicate to browsers whether it is safe to access restricted resources from a separate domain. By default, cross-domain requests are forbidden by the same-origin security policy. The same-origin policy protects browser users from unknowingly sharing session information with bad actors.

The same-origin policy means that a web page served from www.example.com could not, by default, make a call to example.com's APIs at api.example.com because the host name is different. CORS can be used to allow this kind of cross-origin access.

You need CORS in the retail API because the developer portal has a domain name of "*.apigee.io," and the API is accessed via a different domain, "*.apigee.net." In order to invoke the API from the documentation, you will add CORS headers to all API responses, including error responses.

CORS also uses preflight requests. The browser sends a preflight request using the OPTIONS verb to find out whether the next call will be allowed.

Apigee provides a CORS policy that sets all of the required CORS security policies for the API proxy.

For more information about CORS, refer to the Apigee CORS documentation.

Add CORS 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 12.

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

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

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

7. Specify the following values:

Property	**Value**
Name	**CORS-AddCORS**
Display name	**CORS-AddCORS**

8. Click **Add**.

9. Click **Policies > CORS-AddCORS**.

   The CORS configuration includes the following settings:

   AllowOrigins lists the allowed origins. The default configuration allows any Origin, because it sets the allowed origin equal to the Origin that is passed in the request. In a typical production use case, you might only allow requests from specific hostnames.

   AllowMethods specifies the methods that should be allowed for the API.

   AllowHeaders lists the headers that may be passed in the request.

   ExposeHeaders specifies the headers in the response that should be allowed when being called with an Origin. With the default value of *****, no response headers will be stripped from the response.

   MaxAge specifies how long a preflight response may be cached by a browser, in seconds.

   AllowCredentials indicates whether Authorization headers, TLS client certificates, or cookies can be sent in the request.

   GeneratePreflightResponse specifies whether preflight requests with the *OPTIONS* method will be handled.

10. Replace the **AllowMethods** element with:

```
<AllowMethods>GET, PUT, POST, DELETE, PATCH</AllowMethods>
```

content_c

The retail API uses the PATCH method.

11. Replace the **AllowHeaders** element with:

```
<AllowHeaders>origin, x-requested-with, accept, content-type, api
```

content_c

The retail API uses the `apikey` header to specify an API key, so that header must be added to be called from a browser.

12. Replace the **MaxAge** element with:

```
<MaxAge>-1</MaxAge>
```

content_c

This disables the browser caching of the preflight response, so that you will always see the preflight request. In a production use case, you should typically allow caching of the response to avoid making two calls for every request.

13. Replace the **AllowCredentials** element with:

```
<AllowCredentials>true</AllowCredentials>
```

content_c

The retail API requires an Authorization header for creating an order.

The CORS policy configuration should now look like this:

```
<CORS continueOnError="false" enabled="true" name="CORS-
AddCORS">
  <DisplayName>CORS-AddCORS</DisplayName>
  <AllowOrigins>{request.header.origin}</AllowOrigins>
  <AllowMethods>GET, PUT, POST, DELETE, PATCH</AllowMethods>
  <AllowHeaders>origin, x-requested-with, accept, content-type,
```

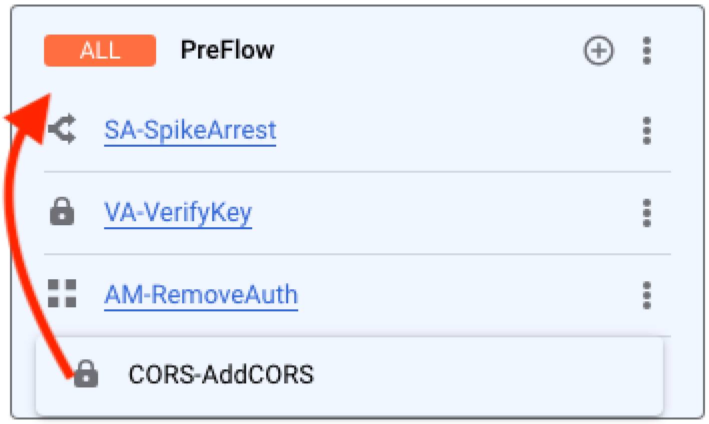
content_c

```
apikey</AllowHeaders>
  <ExposeHeaders>*</ExposeHeaders>
  <MaxAge>-1</MaxAge>
  <AllowCredentials>true</AllowCredentials>
  <GeneratePreflightResponse>true</GeneratePreflightResponse>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</CORS>
```

Note: This implementation sends back the request's Origin in the response via the Access-Control-Allow-Origin header. Any Origin will be allowed, which is not a recommended solution for a production-quality API. A better solution is to only accept requests that come from an origin that has been specifically allowed.

14. In the Navigator menu, click **Proxy endpoints** > **default** > **PreFlow**.

15. Drag the **CORS-AddCORS** policy to execute **first** in the proxy endpoint PreFlow request:



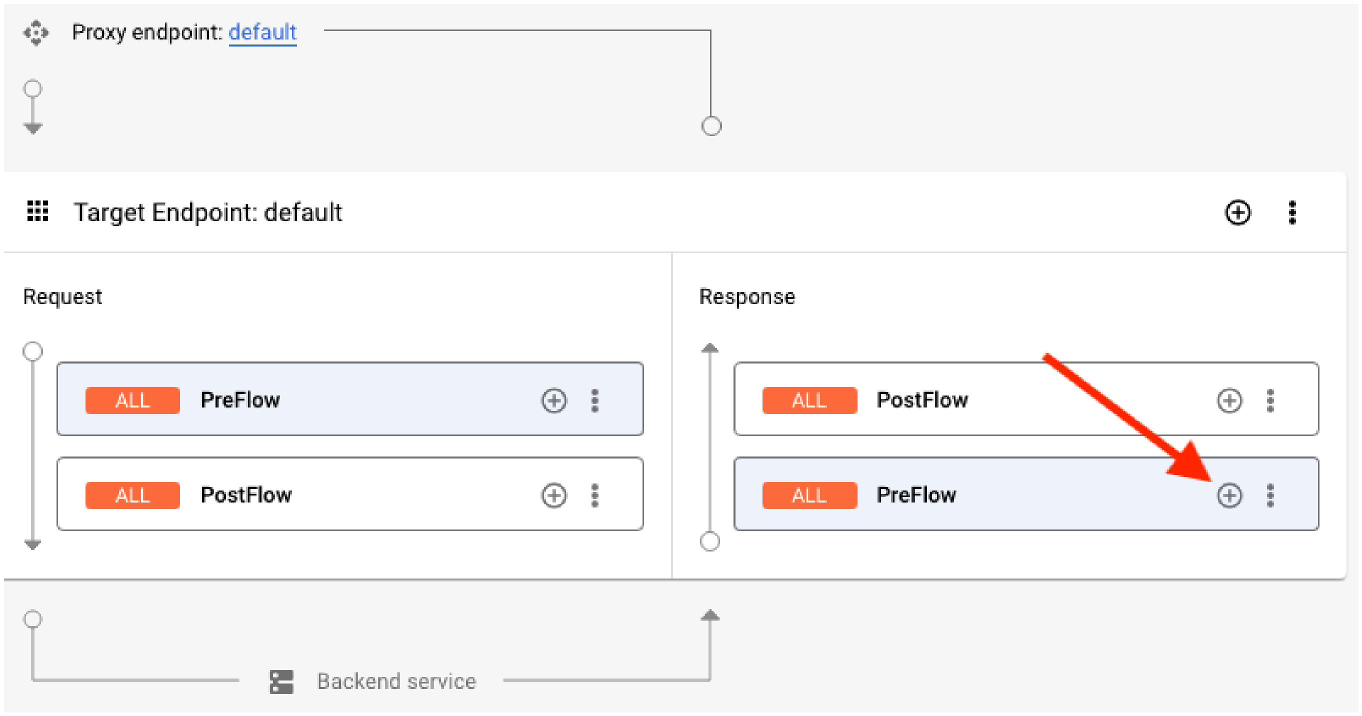
Remove target CORS

You may have noticed during previous labs that the target always returns **Access-Control-Allow-Origin** with a value of "*". You will remove this header from the backend response to ensure that the retail API's CORS implementation is used.

Note: It is a best practice to remove all headers that are not needed from the backend service response. In this lab, we are only removing the Access-Control-Allow-Origin header.

1. In the Navigator menu, click **Target endpoints > default > PreFlow**.

2. On the **Response PreFlow flow**, click **Add Policy Step (+)**.



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

4. Specify the following values:

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

5. Click **Add**.

6. Click **Policies > AM-StripTargetCORS**.

7. Replace the AssignMessage policy configuration with:

```
<AssignMessage continueOnError="false" enabled="true" name="AM-
StripTargetCORS">
  <Remove>
    <Headers>
```

content_c

```
      <Header name="Access-Control-Allow-Origin"/>
    </Headers>
  </Remove>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

8. Click **Save**, and then click **Save as New Revision**.

9. Click **Deploy**.

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

11. Click **Confirm**.

Task 2. Create an integrated developer portal

In this task, you create an integrated developer portal for your Apigee organization.

1. In the Navigator pane, click **Distribution** > **Portals**.

   Note: The portals functionality is not currently accessible from the Google Cloud console. This lab will use the classic Apigee UI for creating the portal.

2. If Portals is not yet implmented in the Cloud Console, Click **Go to Classic Apigee UI**.

   The classic UI opens in a new tab.

3. Click **Get started**, or click **+Portal** if you already have a portal.

4. Enter a name for the portal (**retail**), and then click **Create**.

   Creation may take a minute, and then the retail portal management page should open.

5. If a message to "Enroll in beta for team and audience management features" is displayed, click **Enroll**.

6. To open the new portal in a new tab, click **Live Portal**.

Task 3. Modify your OpenAPI specification

In this task, you modify the OpenAPI specification to match your current proxy implementation.

Download the OpenAPI specification

- In **Cloud Shell**, to download the OpenAPI specification for the backend service, run this curl command:

```
curl https://storage.googleapis.com/cloud-training-cors/developing-
apis/specs/retail-backend.yaml?$(date +%s) --output ~/retail-
backend.yaml
```

content_c

  This curl command downloads a file named *retail-backend.yaml* and stores it in a file with the same name in the home directory.

  Note: `?$(date +%s)` adds a query parameter to the URL that is a string representation of the current date/time. This dynamically changing variable changes the URL and forces curl to retrieve the latest version of a file, even if a previous version is cached.

View the OpenAPI specification in Cloud Shell Editor

1. In Cloud Shell, click **Open Editor**.

2. In the Cloud Shell Editor, click **Open file...**, and then click **retail-backend.yaml**.

   Two changes must be made:

   - **Basic Authentication for updateProductById** must be removed. This is a feature of the backend service, not the retail-v1 proxy.

   - **API key support** must be added for all calls.

   For OpenAPI version 3 specifications, defining security requires two steps:

   - Define the security scheme

   - Attach the security scheme to the resources

   Note: If you make a mistake and cannot figure out how to recover, you can start again by redownloading the retail-backend.yaml file. YAML requires specific spacing and indentation to work. Each indentation level is two spaces, and tabs are not allowed. The following changes should have the correct spacing if you keep the spacing and replace entire lines.

Update the specification

1. Find the **securitySchemes** section on **line 316**. The section currently looks like this:

```
securitySchemes:
  basicAuth:
    type: http
    description: basic auth
    scheme: basic
```

The section defines a scheme named **basicAuth** that uses the basic authentication scheme. This is no longer required because you only allow API key requests.

2. Replace the basicAuth scheme with an API key scheme by replacing **lines 316-320** with:

    ```
    securitySchemes:
      apikey:
        type: apiKey
        in: header
        name: apikey
    ```

    content_c

    The **in** and **name** fields indicate that the API key is found in a **header** named **apikey**. Remember to match the indentation of the document.

3. Remove the basic auth security defined for the updateProductById operation. The **security** field should be on **line 224**.

    ```
          security:
            - basicAuth: []
    ```

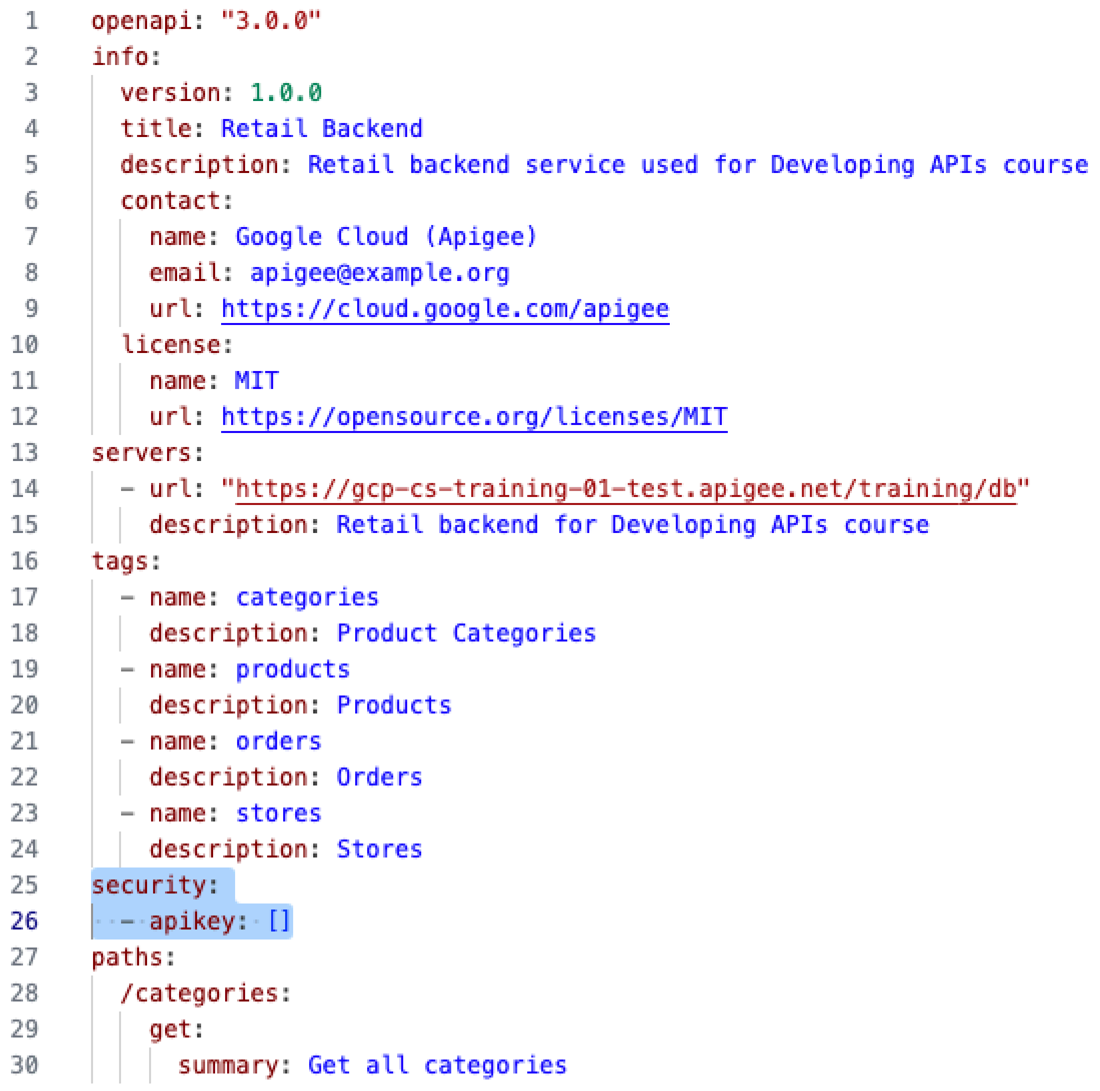
    Remove these two lines (224 and 225) without changing the indentation of surrounding lines.

4. Add API key validation for **all resources**. Insert it just after the **tags** section and just before the **paths** section, starting at **line 25**:

    ```
    security:
      - apikey: []
    ```

    content_c

    Inserting this security setting at the top level will cause it to be applied to all operations. When inserted, it should look like this:

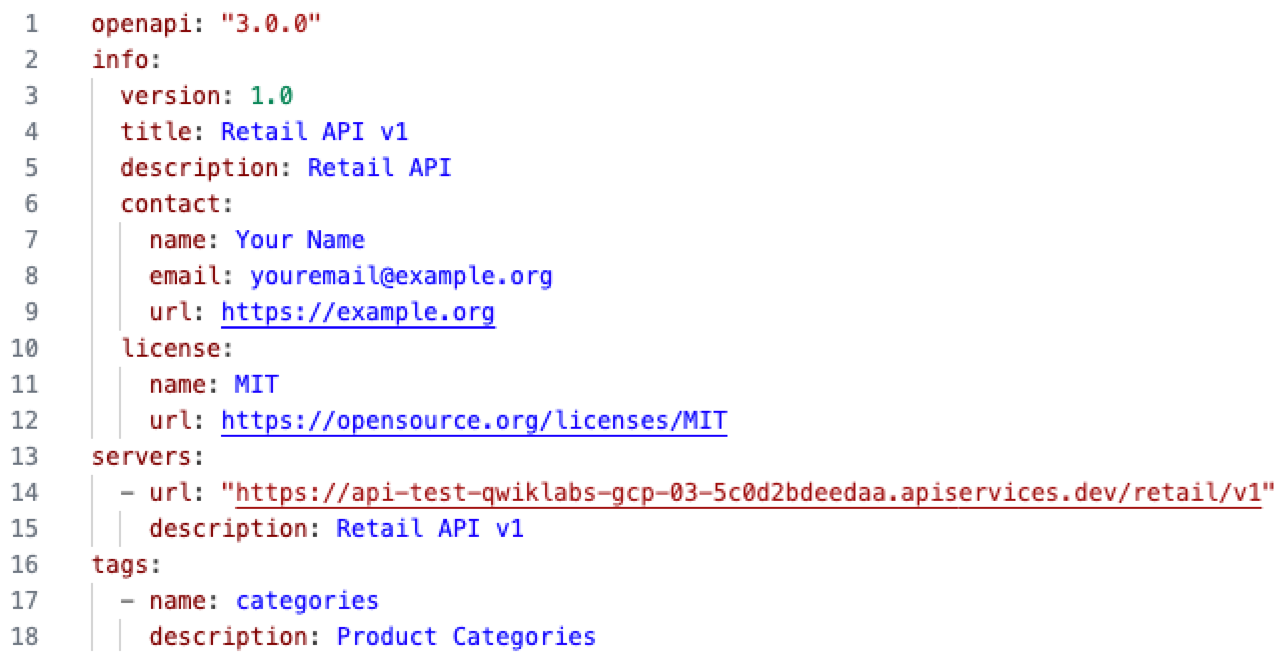

```
openapi: "3.0.0"
info:
  version: 1.0.0
  title: Retail Backend
  description: Retail backend service used for Developing APIs course
  contact:
    name: Google Cloud (Apigee)
    email: apigee@example.org
    url: https://cloud.google.com/apigee
  license:
    name: MIT
    url: https://opensource.org/licenses/MIT
servers:
  - url: "https://gcp-cs-training-01-test.apigee.net/training/db"
    description: Retail backend for Developing APIs course
tags:
  - name: categories
    description: Product Categories
  - name: products
    description: Products
  - name: orders
    description: Orders
  - name: stores
    description: Stores
security:
  - apikey: []
paths:
  /categories:
    get:
      summary: Get all categories
```

5. Update the top 15 lines to reference your API proxy instead of the backend service. Replace **lines 1-15** with:

```
openapi: "3.0.0"                                                    content_c
info:
  version: 1.0
  title: Retail API v1
  description: Retail API
  contact:
    name: Your Name
    email: youremail@example.org
    url: https://example.org
  license:
    name: MIT
    url: https://opensource.org/licenses/MIT
servers:
  - url: "https://api-test-qwiklabs-gcp-03-
```

```
f6e0f95a7e4e.apiservices.dev/retail/v1"
    description: Retail API v1
```

The top of your specification should resemble this, with a different Google Cloud Project name:



```yaml
openapi: "3.0.0"
info:
  version: 1.0
  title: Retail API v1
  description: Retail API
  contact:
    name: Your Name
    email: youremail@example.org
    url: https://example.org
  license:
    name: MIT
    url: https://opensource.org/licenses/MIT
servers:
  - url: "https://api-test-qwiklabs-gcp-03-5c0d2bdeedaa.apiservices.dev/retail/v1"
    description: Retail API v1
tags:
  - name: categories
    description: Product Categories
```

6. In the IDE menu (☰), click **File** > **Save As...**.

7. For the filename, replace `retail-backend` with `retail-v1`, and then click **OK**.

8. Click **Open Terminal**.

9. Select Cloud Shell's *More* menu (⋮), and then click **Download**.

10. Enter `retail-v1.yaml` and then click **Download**.

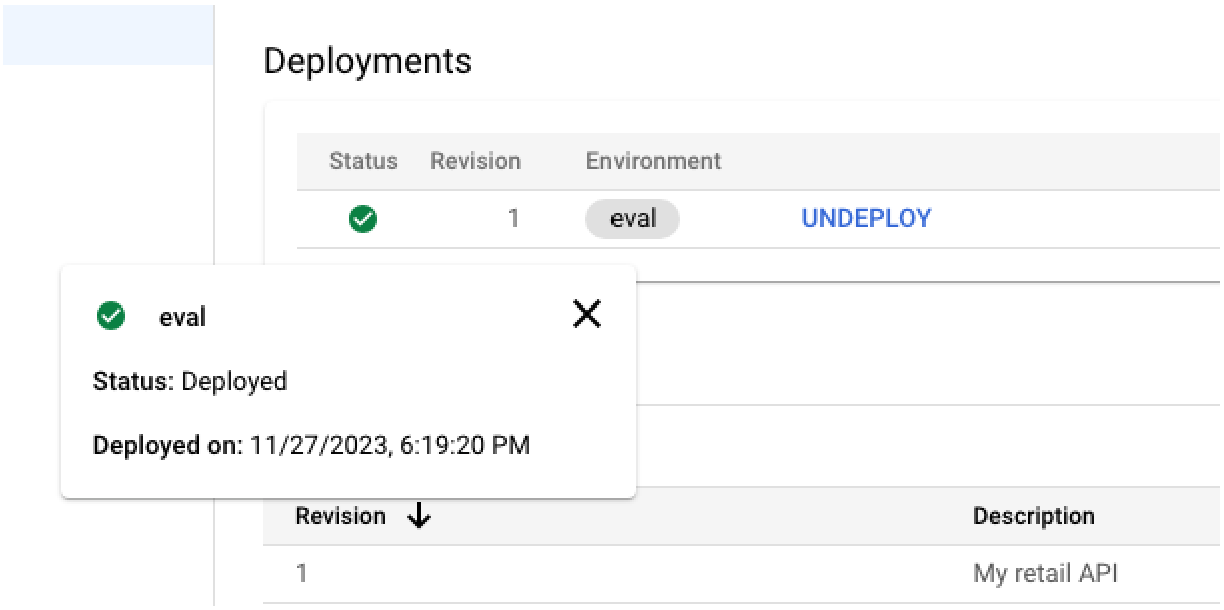
    This will download the file to your local machine. You will use the updated specification with the developer portal.

11. Close Cloud Shell.

Task 4. Test the CORS functionality

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

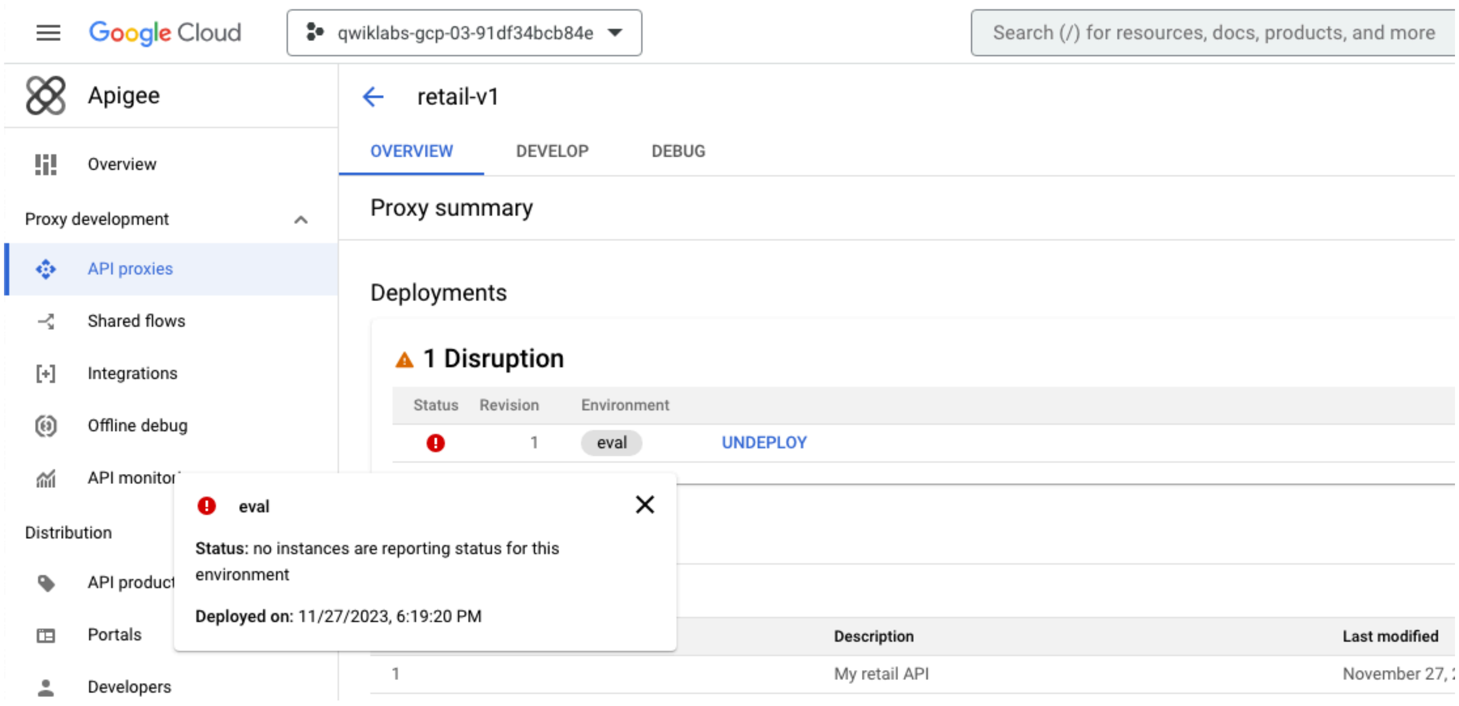
eval

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 may see a red warning sign. Hold the pointer over the **Status** icon 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

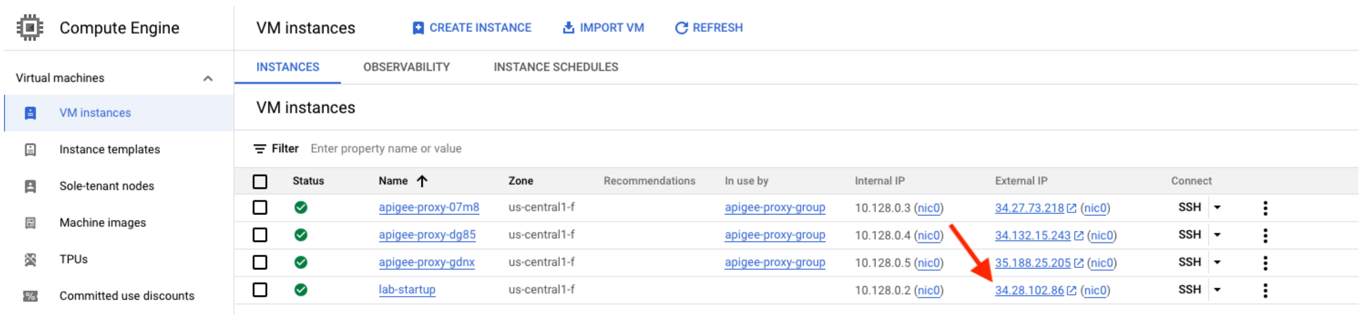
	Description	Last modified
1	My retail API	November 27,

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.

Check provisioning dashboard

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.



Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	apigee-proxy-07m8	us-central1-f		apigee-proxy-group	10.128.0.3 (nic0)	34.27.73.218 (nic0)	SSH
✓	apigee-proxy-dg85	us-central1-f		apigee-proxy-group	10.128.0.4 (nic0)	34.132.15.243 (nic0)	SSH
✓	apigee-proxy-gdnx	us-central1-f		apigee-proxy-group	10.128.0.5 (nic0)	35.188.25.205 (nic0)	SSH
✓	lab-startup	us-central1-f			10.128.0.2 (nic0)	34.28.102.86 (nic0)	SSH

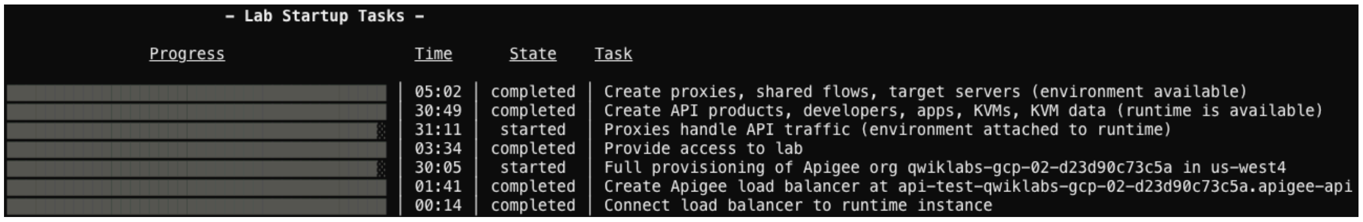
3. If you see a redirect notice page, click the link to the external IP address.

   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
```

In this case, you need to wait for *Proxies handle API traffic* to complete.

While you are waiting

Read:

- OpenAPI specification
- Building your integrated portal
- Managing API products

Start a debug session

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 **Debug** tab, and then click **Start Debug Session**.

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

5. Click **Start**.

Store the app's key in a shell variable

The API key may be retrieved directly from the app accessible on the **Publish** > **Apps** page. It can also be retrieved via Apigee API call.

- In **Cloud Shell**, run the following command:

```
export API_KEY=$(curl -q -s -H "Authorization: Bearer $(gcloud auth
print-access-token)" -X GET
"https://apigee.googleapis.com/v1/organizations/${GOOGLE_CLOUD_PROJECT
app" | jq --raw-output '.credentials[0].consumerKey'); echo "export
API_KEY=${API_KEY}" >> ~/.profile; echo "API_KEY=${API_KEY}"
```

content_c

This command retrieves a Google Cloud access token for the logged-in user, sending it as a Bearer token to the Apigee API call. It retrieves the **retail-app** app details as a JSON response, which is parsed by **jq** to retrieve the app's key. That key is then put into the **API_KEY** environment variable, and the export command is concatenated onto the **.profile** file which runs automatically when starting a Cloud Shell tab.

Note: If you run the command and it shows API_KEY=null, the runtime instance is probably not yet available.

Test the CORS functionality

1. Open Cloud Shell, and then drag the Cloud Shell pane so you can see both Cloud Shell and the Debug session.

2. Make a request to retrieve a single category:

```
curl -i -H "apikey: ${API_KEY}" -X GET "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories/1
```

content_co

   - There is no Origin, so the **Access-Control-Allow-Origin** header is not returned.

Note: If the call returns an Access-Control-Allow-Origin header with the value of "*", your proxy is not correctly stripping the target CORS from the target response.

3. Make another request, but include the Origin header this time. This tests a normal CORS request:

```
curl -i -H "Origin: https://www.example.com" -H "apikey:
${API_KEY}" -X GET "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/categories/1
```

content_co

- The **Access-Control-Allow-Origin** header should be returned with the Origin value you supplied in the request, and the **Access-Control-Expose-Headers** and **Access-Control-Allow-Credentials** headers are also returned.

4. Make a 404 Not Found request to test a fault:

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

content_co

The CORS headers are correctly returned even though a fault was raised and the Response flows were not executed.

5. Make a preflight request:

```
curl -i -H "Origin: https://www.example.com" -H "Access-Control-
Request-Method: PATCH" -H "Access-Control-Request-Headers:
Accept,Content-Type,apikey" -H "apikey: ${API_KEY}" -X OPTIONS
"https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/retail/v1/products/136
```

content_co

The CORS preflight headers are correctly set in the response.

Task 5. Publish an API to the developer portal

In this task you associate an API product with an OpenAPI specification and publish them to the developer portal.

1. Return to the portal tab, and then click **API catalog**.

2. Click +.

3. Select the **retail (full access)** API product, and click **Next**.

4. Customize the API details:

Property	**Value**
Published (listed in the catalog)	*selected*
Display title	**Retail**
Display description	**Retail API v1**
API visibility	*select* Public (visible to anyone)

- *Published* makes the API visible on the portal.
- *Public* allows APIs to be seen even if the user is not logged in to the portal.

5. Click **Select image** and then click the **Image URL** link.

6. Set the image URL to:

```
https://storage.googleapis.com/cloud-training/developing-apis/images/image-shopping.png
```

content_c

7. Click **Select**.

8. In the **API documentation** section, select **OpenAPI document**, and then click **Select Document**.

9. Click on the cloud image to select a file to upload.

10. Select the OpenAPI spec file you downloaded from Cloud Shell (retail-v1.yaml), and then click **Open**.

11. Click **Select**.

12. Click **Save**.

Task 6. Test the API using the developer portal

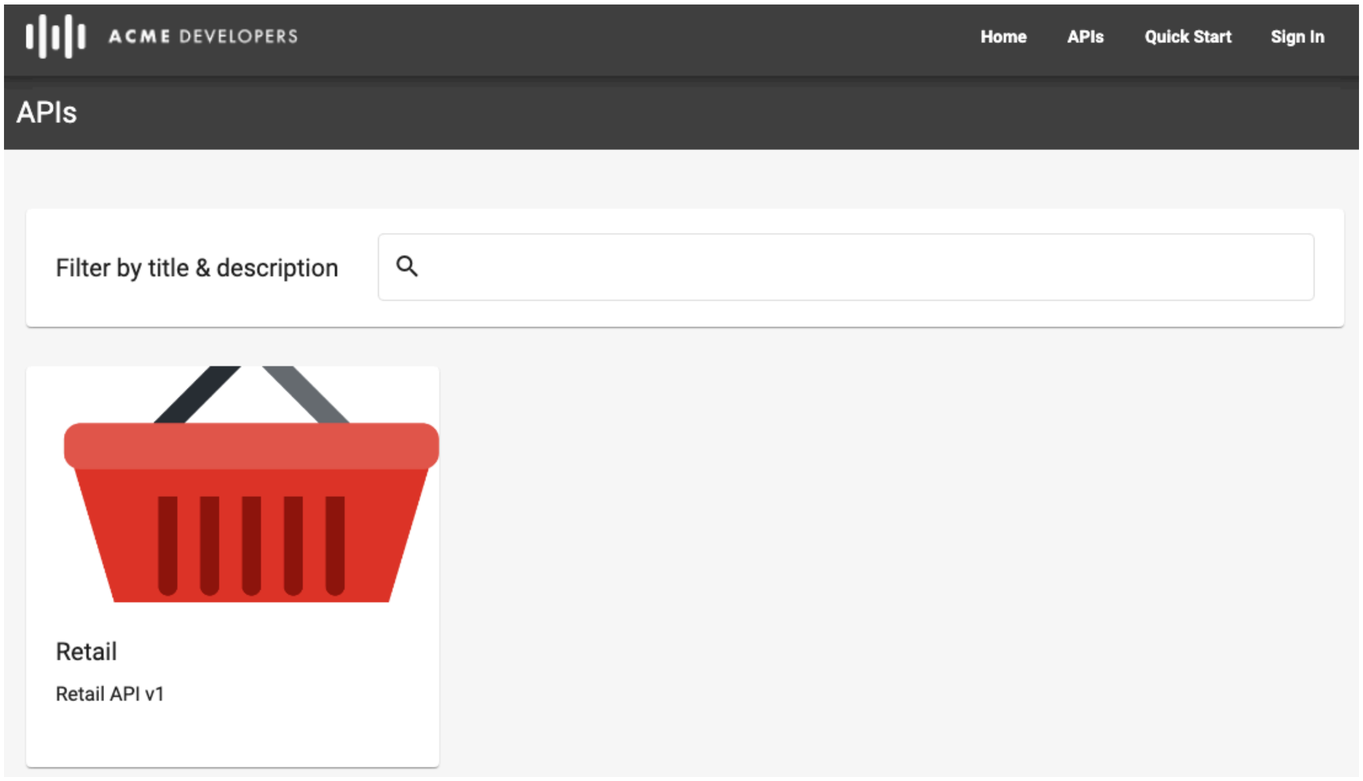
In this task you use the developer portal and make requests to your API.

Start a new debug session

1. Return to the Cloud Console tab.
2. Click the **Debug** tab, and then click **Start Debug Session**.
3. In the **Start debug session** pane, on the Environment dropdown, select **eval**.
4. Click **Start**.

Test in the developer portal

1. Return to the **Live Portal** tab.
2. Click **APIs** in the top menu bar.
3. Refresh the APIs page to see the new API:



4. Click on the **Retail** API.

5. Click **Authorize**.

6. In Cloud Shell, to get the API key, run the following command:

```
echo ${API_KEY}
```

content_c

> **Note:** If your API key does not show when you use the command, return to task 4 to populate the key in a shell variable.

7. Copy the API key from Cloud Shell.

8. In the dialog box, paste the API key, and then click **Authorize**.

9. Click **OK** to close the dialog box.

10. In the **Paths** menu on the left, select **/categories GET** in the left navigation bar, and go to the **Try this API** section.

11. Click **Execute**.

    You should see a successful response.

12. In the **Paths** menu, click **/products GET**.

13. Click **Try it out**, and then click **Execute**. You should see a 200 response, with the list of products in the **Response body** box.

14. Copy a random product ID from the response, and note the current overall rating.

    > **Note:** Note the rating in the GET /products call. Do not GET the product by ID, or you will cache the product and will not be able to see updates you make to the product.

15. In the left navigation bar, select **/products/**, and then select **{productId} PATCH**.

16. In the **Try this API** section, enter the product ID you chose from the **GET /products** call, set a new **overall_rating** in the request body, and click **Execute**.

    You should see two new requests in the debug tool. The first is the CORS preflight with the **OPTIONS** verb. The proxy returns 200 OK with the CORS headers. The second is the **PATCH** request.

    If you repeat the same call, you will see both the OPTIONS and PATCH requests again because the **Access-Control-Max-Age** header is not allowing the browser to cache the OPTIONS response.

    > **Note:** If an error message says "TypeError: Failed to fetch", your API's CORS implementation is probably broken. You can use the browser Developer Tools to debug the issue.

17. In the left navigation bar, select **/products/**, and then select **{productId} GET**.

18. In the **Try this API** section, **enter the same product ID**, and click **Execute**. You should see that the response contains the new overall rating.

Note: The responses to "GET /products/{productId}" are cached. If your testing seems to indicate that the PATCH command is not working, it may be pulling the previous response from the cache. Choose a different product and try again. Remember, do not perform a "GET /products/{productId}" before you have updated the overall_rating.

Task 7. Sign up as an app developer

In this task, you use the developer portal to sign up as an app developer and get an API key.

Create a new developer account

1. In the upper-right corner of the portal, click **Sign In**.
2. Then click the **Create an account** link.
3. Enter your information. The email address must be a valid email account because you will need to open an email to validate the account.
4. Click **Create Account**.

   An email message should be received in your email box shortly.

5. When you receive the email, click on the registration link. It must be clicked within 10 minutes.
6. Click on **Sign in** in the portal and login using your email address and password.
7. Click on your email address in the upper-right corner, and then click **Apps**.

Create an application

1. Click **+New App**. Use `my-retail-app` for your app's name, and optionally add a description.

2. Click **Enable** to enable the Retail API.

   This will add the *Retail* API product to the app.

3. Click **Save**.

   A key and secret are created, and you can copy the API key into your clipboard and start using it.

Test the API key

1. Click **APIs** at the top of the portal.

2. Then click **Retail** to return to the Retail API page.

3. Click **Authorize**.

4. In the **Please select app** dropdown, select your **my-retail-app** to automatically use its API key, or to manually specify the API key, select **Manually enter key**.

5. Click **OK**.

   You can continue to test your API in the developer portal. Your API key can also be used from curl or a REST client.

Verify the app in the Apigee console

1. Return to the Apigee console, and click **Distribution > Developers**.

   Your newly created developer will appear in the list of developers.

2. Click **Distribution > Apps**.

Your newly created app will appear in the list of apps.

Congratulations!

In this lab, you learned how to add CORS to your proxy, how to modify your OpenAPI specification for API key use, and how to create, configure, and use an integrated developer portal.

End your lab