

Qlog: Build and Manage APIs with Apigee: Challenge Lab

13 Sep 2020

In this article, we will go through the lab **GSP336 Build and Manage APIs with Apigee: Challenge Lab**, which is labeled as an expert-level exercise. You will practice the skills and knowledge in the Build and Manage APIs with Apigee.

Topics tested:

- Create an API Facade and add functionality
- Share the APIs with partners through a developer portal
- Route traffic to different backend implementations of the API

The challenge contains 4 required tasks:

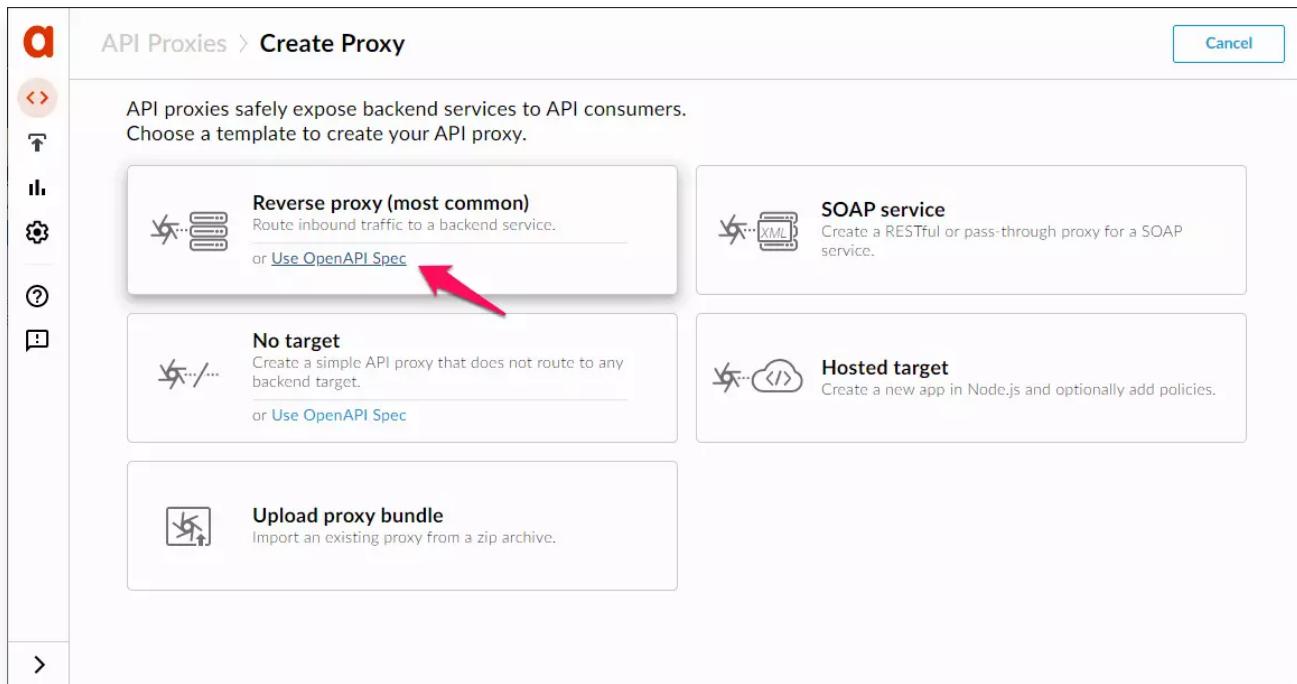
The screenshot shows a yellow-themed user interface for a challenge. At the top left, it says "Checkpoints" with an arrow pointing right. Below that, there are four task cards, each with a "Check my progress" button and a score out of 25 or 30.

Task Description	Score / Total
Upload API Proxy bundle to GCS	/ 20
Create a service account with permissions to write logs	/ 25
Add Authentication: API Key verification	/ 25
Route traffic to mock backend from real backend	/ 30

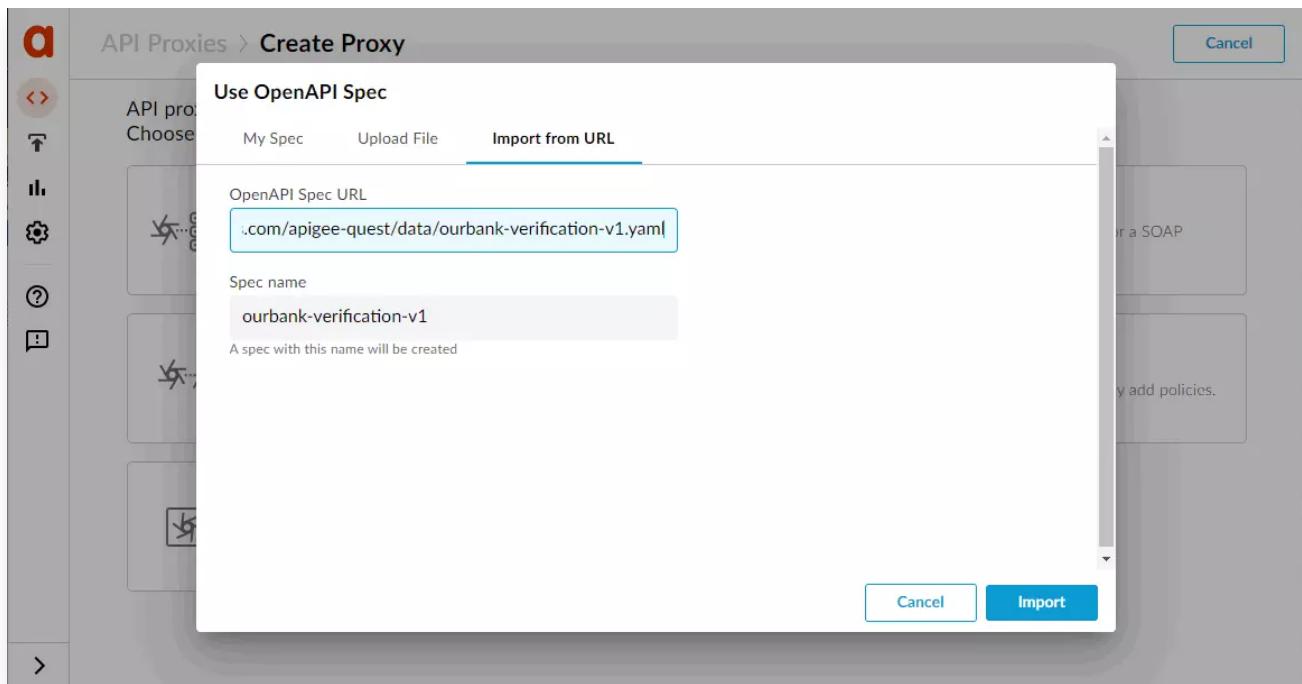
Task 1 - Create API Specification and Generate an API Proxy

Define a RESTful API in Apigee using an API specification

1. Login to your Apigee account (<https://login.apigee.com/login>)
2. In the Apigee console, select **Develop > API Proxies** from the left pane.
3. Click on the **+Proxy** button to create a new proxy.
4. Create a **Reverse proxy** by clicking “Use OpenAPI Spec” as shown in the picture below.



5. In the *Use OpenAPI Spec* dialog, select **Import from URL** and enter the following values:
 - **OpenAPI Spec URL:** <https://storage.googleapis.com/apigee-quest/data/ourbank-verification-v1.yaml>
 - **Spec name:** e.g. `ourbank-verification-v1`



6. Click **Import**.

7. Find the “View JSON response” endpoint URL of Mock Target API from [this Apigee Doc page](#), which is:

<https://mocktarget.apigee.net/json>

8. Copy the URL to the field **Target (Existing API)**, then click **Next**.

Proxy details

Name: **Verification-API-v1** Available

Base path: **/verification-api-v1**
This proxy will handle requests on hostname/base-path [Learn more](#)

Description:
This is OurBank's verification API. There are two operations available. You can verify credit card numbers and you can verify addresses.

Target (Existing API):
https://mocktarget.apigee.net/json
The URL of the backend service that this proxy invokes

Next

9. Check **Add CORS header** to enable CORS headers in Apigee, then click **Next**.

Common policies

Security: Authorization
 API Key
 OAuth 2.0
 Pass through (no authorization)

Quota
 Impose quotas per App
Available only for proxies with authorization.
Quota details are configured in API Products. [Learn more](#)

Security: Browser
 Add CORS headers
Required for enabling web browser access to this proxy.
[Learn more](#)

Next

10. Click **Next** to continue.

OpenAPI operations

Select the operations to generate condition flows. You can update the operations later. [Learn more](#)

PATH	VERB	OPERATION	SUMMARY
/verifyCard	POST	verifyCreditCard	Verify a credit card
/verifyAddress	POST	verifyAddress	Verify an address

11. Check the box next to **default**, then click **Next**.

Virtual hosts

For incoming traffic, select the virtual hosts this proxy will bind to when it is deployed. [Learn more](#)

VIRTUAL HOST	HOST ALIAS
secure	https://[REDACTED].prod.apigee.net (Environment: prod) https://[REDACTED]-test.apigee.net (Environment: test)
default	http://[REDACTED].prod.apigee.net (Environment: prod) http://[REDACTED]-test.apigee.net (Environment: test)

12. Check the box next to **default**, then click **Create and deploy**.

The screenshot shows the 'Summary' tab of an API proxy configuration. The proxy name is 'Verification-API-v1', it is a Reverse Proxy, and its base path is '/verification-api-v1'. The target is 'https://mocktarget.apigee.net/json'. Policies include 'Authorization - Pass through (no authorization), CORS header'. The OpenAPI Spec is 'Verification API (v1)'. Virtual hosts are 'secure, default'. Deployment is set to 'test' environment.

Summary	
Proxy name	Verification-API-v1
Proxy type	Reverse Proxy
Proxy base path	/verification-api-v1
Target	https://mocktarget.apigee.net/json
Policies	Authorization - Pass through (no authorization), CORS header
OpenAPI Spec	Verification API (v1)
Virtual hosts	secure, default

Optional Deployment

prod test

The proxy will begin handling requests only after it's deployed to an environment.

Apigee now deploys the API proxy into your test environment. Click **Edit proxy** to view the deployed proxy.

Provision a Mock Response in Apigee

1. In the page of **API Proxies > Verification-API-v1**, click the **Develop** tab in the top right.
2. To add a policy to your proxy, click on **Proxy Endpoints → PreFlow** in the Navigator tab, then in the Response pipeline, click **+ Step** to add a step.

The screenshot shows the Apigee API Proxy Editor interface. On the left, the Navigator pane lists 'Verification-API-v11' with sections for Policies, Proxy Endpoints, Target Endpoints, and Resources. Under 'Proxy Endpoints', there is a 'default' section containing 'All' and 'PreFlow'. The 'PreFlow' step is selected, and its configuration is shown in the main panel. The configuration XML includes logic for 'verifyCreditCard' and 'verifyAddress' using POST methods. The right side of the screen shows the 'Property Inspector' for the 'PreFlow' step, which has a 'name' of 'PreFlow' and is set to 'Request'. At the bottom left, a message says 'Deployed to test'.

3. Select **Assign Message** from the left menu, then click **Add**.

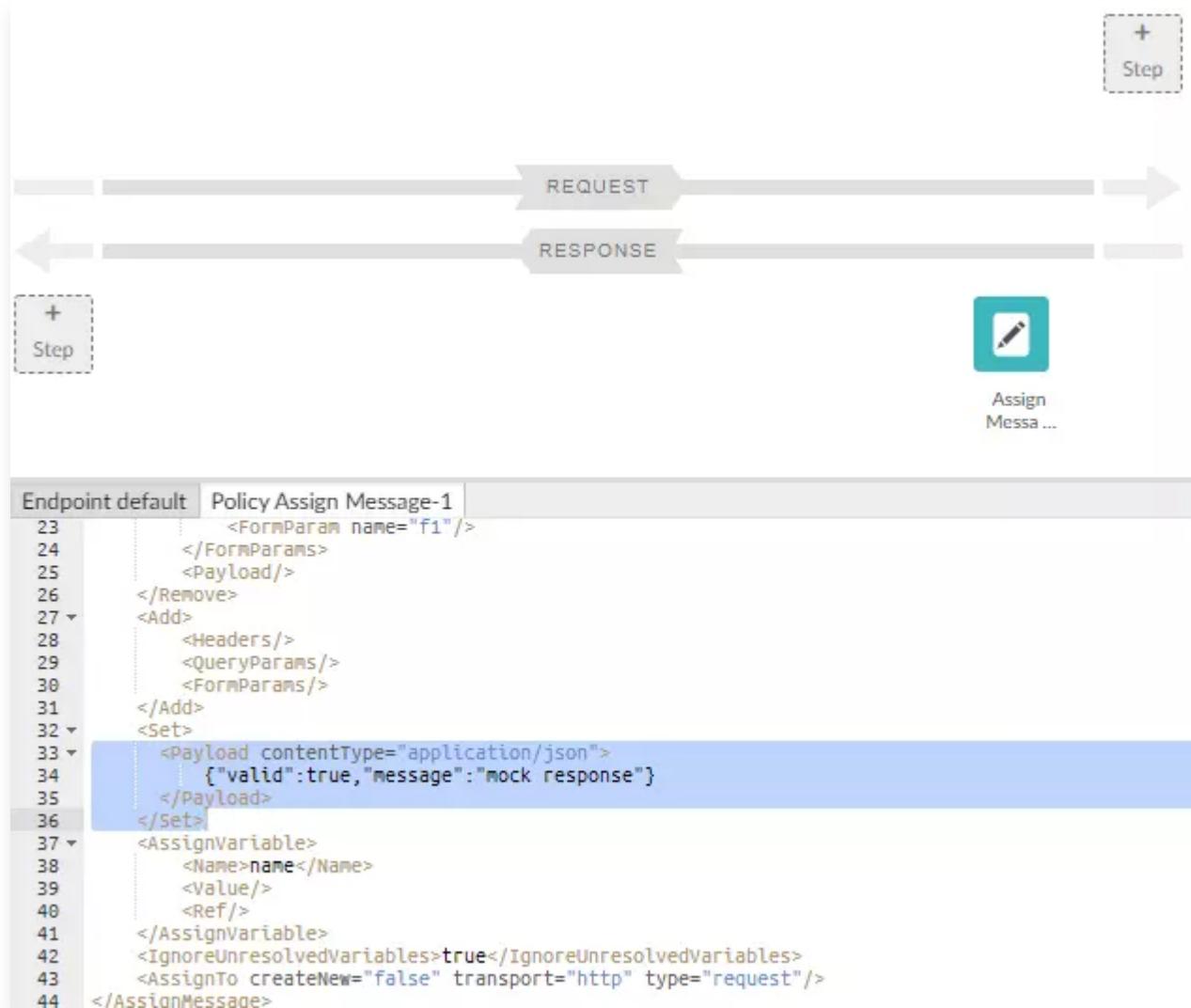
The screenshot shows the 'Add Step' dialog box. On the left, a list of policy types is shown, with 'Assign Message' selected. The right side of the dialog is filled with configuration fields for the selected policy type. It shows 'Policy Type' as 'Assign Message', 'Display Name' as 'Assign Message-1', and 'Name' as 'Assign-Message-1'. At the bottom right, there are 'Cancel' and 'Add' buttons.

4. Modify Policy Assign Message-1:

- Replace the **Set** element in the policy with the below.

```
<Set>
  <Payload contentType="application/json">
    {"valid":true,"message":"mock response"}
  </Payload>
</Set>
```

After the above procedure, the **Proxy Endpoints → PreFlow** should look like the picture below.



Testing

The deployment can be tested using the following curl statement in the Cloud Shell:

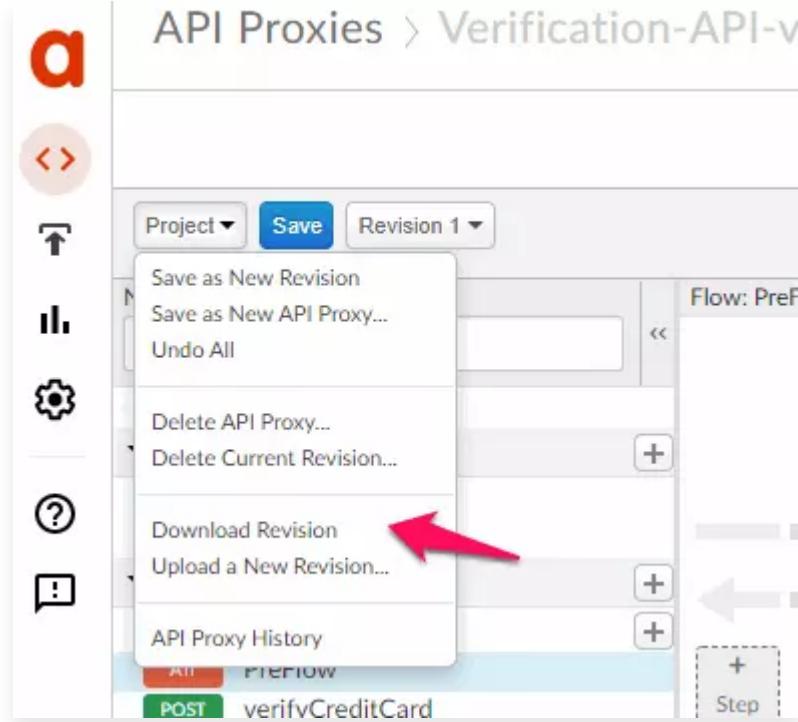
```
export APIGEE_ORG=<YOUR_APIGEE_ORG_NAME>

curl -X POST \
  https://${APIGEE_ORG}-test.apigee.net/verification-api-v1/verifyCard \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: 89236919-eabe-4357-e4c4-079f20ecd798' \
  -d '{
    "number": "2221005276762844",
    "cvv": "345",
    "expiration": "10/2025"
}'
```

Replace <YOUR_APIGEE_ORG_NAME> with your Apigee organization name.

Upload API Proxy bundle to GCS

1. Go back to Apigee, click on **Project > Download Revision** at the top-right of the Deploy tab.



2. Extract the downloaded zip file to your local storage.
3. In the Cloud Console, click on **Navigation Menu > Storage**.
4. Create a new bucket.

5. Upload the `apiproxy` folder to the bucket.

Task 2 - Add Policies to the API Proxy

Create a service account with permissions to write logs

1. In the Cloud Console, click on **Navigation Menu > IAM & admin > Service accounts**.
2. Click **Create Service Account**, then enter the following:
 - Service account name: `apigee-stackdriver`
 - Service account description: `Service account for Apigee Stackdriver integration`
3. Click **Create** to continue.
4. Click into **Select a Role** field and choose **Logging > Logs Writer** permission.
Click **Continue** then click **Done**.
5. After creating service account ‘apigee-stackdriver’, click on three dots at the right corner and click **Create Key** in the dropdown, then **Create** to download your JSON output.

Create an extension and deploy it to test the environment using this service account

1. Go back to Apigee, naviagte to **Admin > Extensions** from the left navigation menu, then **+ Add Extension** to create a new extension.
2. On the new Extension Properties page, click on the **Google Stackdriver Logging** extension.

3. Enter a name (e.g. `stackdriver-logging-extension`) and an optional description for the Extension instance, then click **Create**.
4. On the Extension detail page, click the arrow (>) for the test environment to configure the instance for the Apigee environment.
5. In the configuration dialog, enter the following information:
 - Select the latest extension version from the Version drop-down list.
 - Add your GCP Project ID (which you can get from the Home Console).
 - Open the downloaded JSON file and copy/paste the contents into the Credential field in the Apigee UI.
6. Click **Save**.
7. Once the configuration is saved, click on the **Deploy** button for the Test environment.

The screenshot shows the Apigee Extensions detail page for the 'stackdriver-logging-extension'. The top navigation bar shows 'Extensions > stackdriver-logging-extension'. On the left, there is a sidebar with icons for clone, delete, and other settings. The main area displays the extension details: 'Google Stackdriver Logging' by Google, version v1.5.0, released on 2020-07-01, with a 'View Details' link. Below this is the 'Extension Properties' section, which includes fields for Name (set to 'stackdriver-logging-extension') and Description. The 'Environment Configurations' section shows two environments: 'prod' and 'test'. The 'prod' environment has a 'Deploy' button. The 'test' environment has a green checkmark and the word 'Deployed', with a 'Deploy' button next to it. A large blue arrow icon is visible on the far left of the sidebar.

Create an Extension policy in the PostFlow response path

1. In Apigee, navigate to **Develop > API Proxies** from the left menu.
2. Click to open **Verification-API-v1** from the proxy list.

3. Click the **Develop** tab in the top right.
4. To add a policy to your proxy, click on **Proxy Endpoints** → **PostFlow** in the Navigator tab, then in the *Response* pipeline, click **+ Step** to add a step.

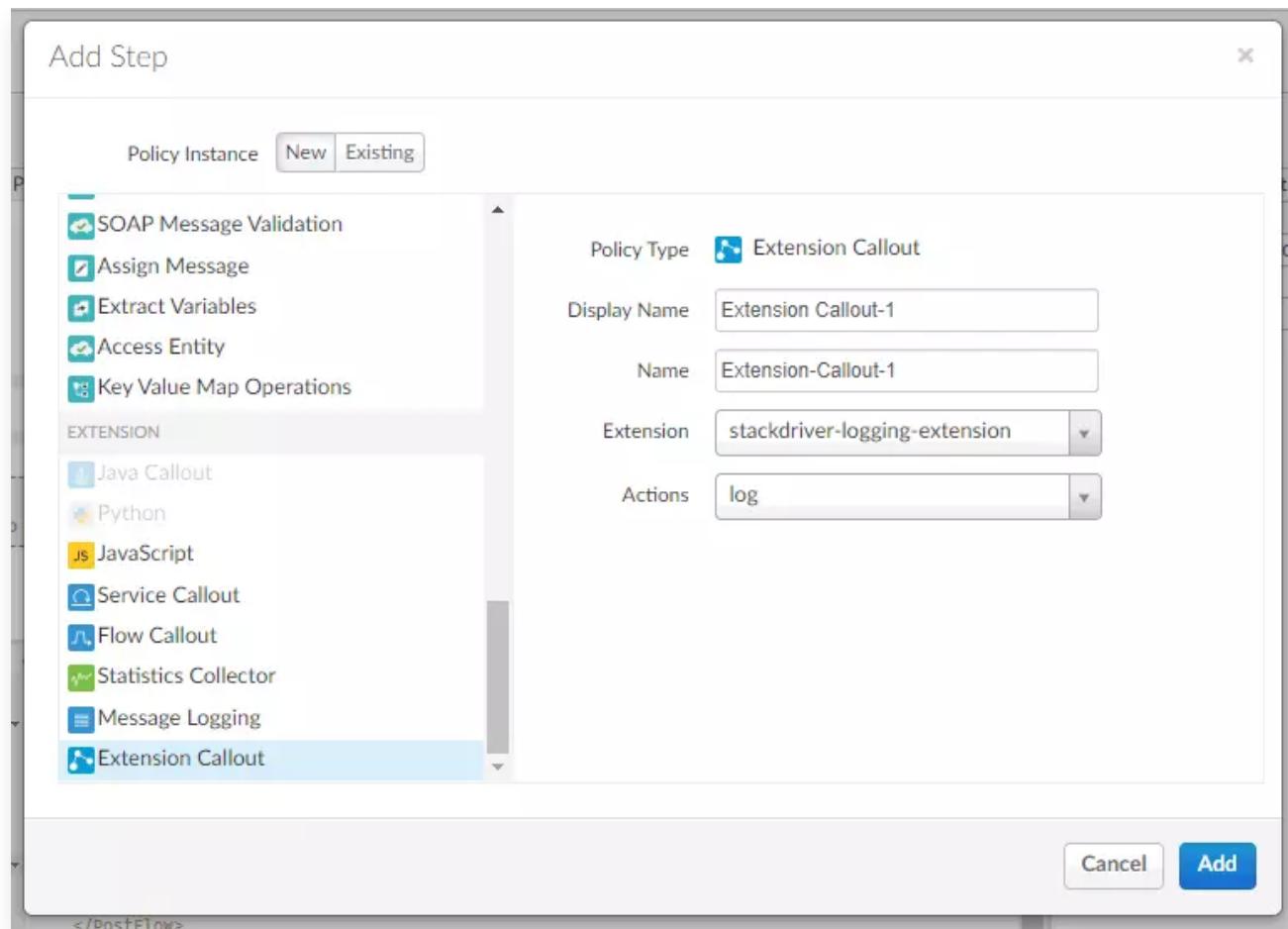
The screenshot shows the Qlog interface with the 'Flow: PostFlow' tab selected. On the left, the 'Navigator' sidebar lists various policies and endpoints. A red arrow points to the 'PostFlow' item under the 'Proxy Endpoints' section. In the main area, the 'Response' pipeline is shown with a 'Step' button highlighted by a red arrow. Below the pipeline, a code editor displays XML configuration for the PostFlow.

```

<Condition>(proxy.pathsuffix MatchesPath "/verifyCa
</Flow>
<Flow name="verifyAddress">
  <Description>Verify an address</Description>
  <Request/>
  <Response/>
  <Condition>(proxy.pathsuffix MatchesPath "/verifyAd
</Flow>
</Flows>

```

5. In the left menu scroll down to the end, then select **Extension Callout**. Select your extension name from the **Extension** dropdown menu. Select **Log** from the Actions dropdown. Click **Add**.

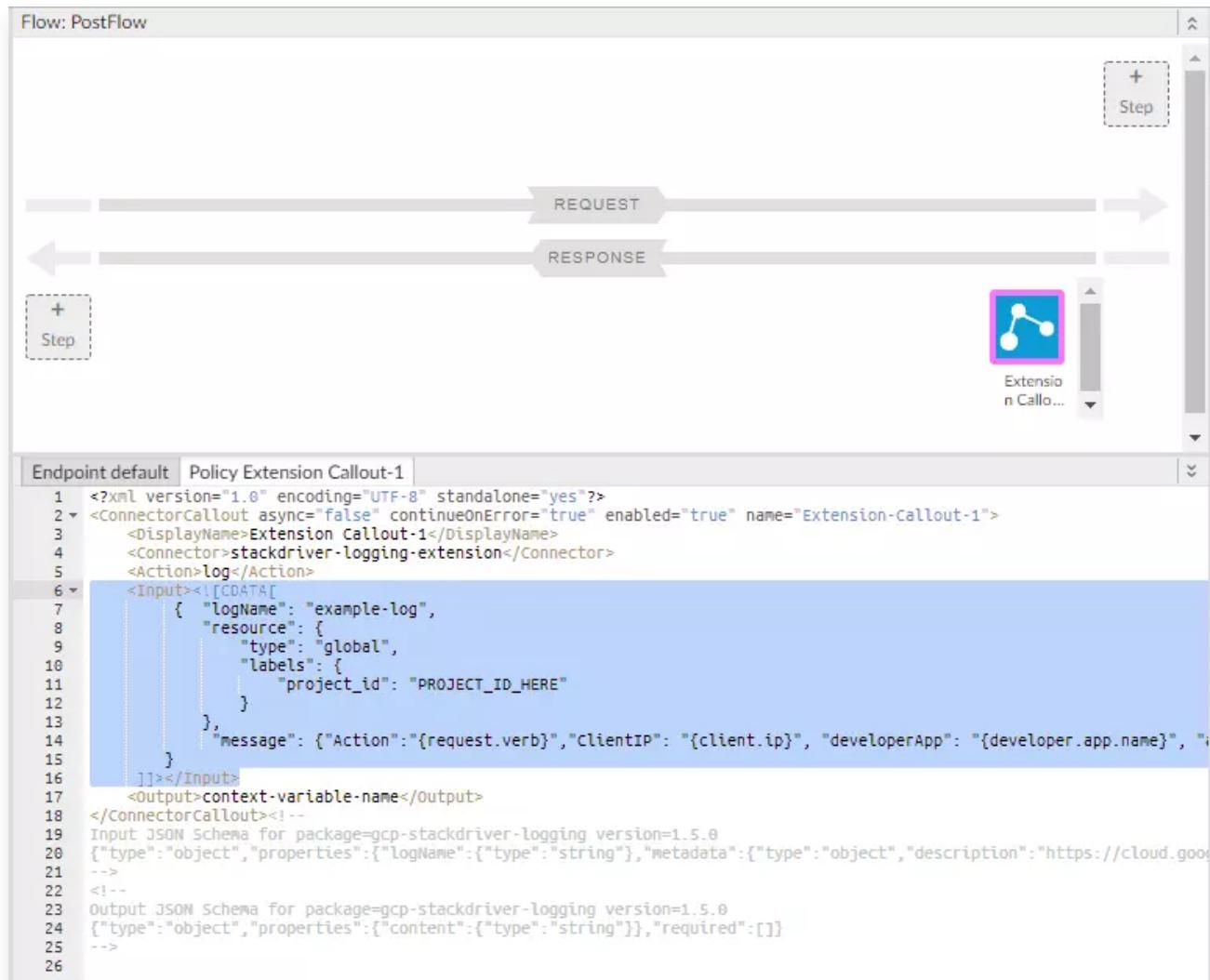


6. Modify Policy Extension Callout-1:

- Replace the **Input** element in the policy with the below (make sure indentation is correct)
- Replace **PROJECT_ID_HERE** with GCP Project ID for this lab.

```
<Input><![CDATA[
  {
    "logName": "example-log",
    "resource": {
      "type": "global",
      "labels": {
        "project_id": "PROJECT_ID_HERE"
      }
    },
    "message": {"Action": "{request.verb}", "clientIP": "{client.ip}",
                "developerApp": "{developer.app.name}",
                "apiKeyParam": "{request.queryparam.apikey}",
                "responsePayload": "{response.content}"}
  }
]></Input>
```

After the above procedure, the **Proxy Endpoints → PostFlow** should look like the picture below.



Add API Key verification

1. To add a policy to your proxy, click on **Proxy Endpoints → PreFlow** in the Navigator tab, then in the *Response* pipeline, click **+ Step** to add a step.

Navigator

Verification-API-v11

- Policies
 - Add CORS
 - Assign Message-1
 - Extension Callout-1
- Proxy Endpoints
 - default
 - All PreFlow
 - POST verifyCreditCard
 - POST verifyAddress
 - All PostFlow
- Target Endpoints
- Resources

Flow: PreFlow

REQUEST → RESPONSE ←

Code default

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <ProxyEndpoint name="default">
3  <Preflow name="PreFlow">
4  <Request/>
5  <Response>
6  <Step>
7  <Name>Assign-Message-1</Name>
8  </Step>
9  </Response>
10 </Preflow>

```

Assign Messa ...

2. Select **Verify API Key** from the left menu, then click **Add**.

Add Step

Policy Instance

Policy Type Verify API Key

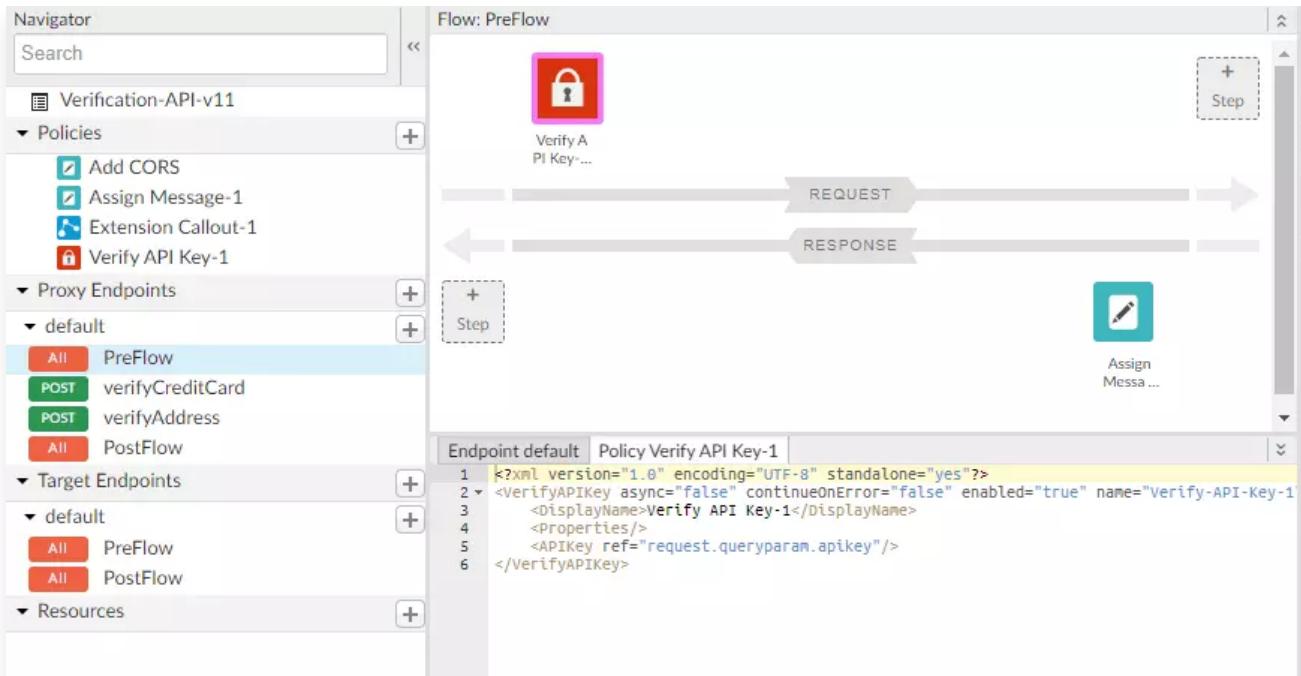
Display Name

Name

Policy Instance

- Delete QLOG v1.0 into
- Verify API Key
- Access Control
- Generate SAML Assertion
- Validate SAML Assertion
- Generate JWT
- Verify JWT
- Decode JWT
- Generate JWS
- Verify JWS
- Decode JWS
- MEDIATION**
- JSON to XML
- XML to JSON
- Raise Fault

After the above procedure, the **Proxy Endpoints → PreFlow** should look like the picture below.



Create an API Product and an App

1. In Apigee, navigate to **Publish > API Products** from the left menu. Then Click **+ API Product**.
2. Create a new API product with the following details:
 - Name:** e.g. `ourBank API`
 - Display Name:** e.g. `ourBank API`
 - Environment:** Check `test`
 - Access:** Select `Public` from the dropdown menu
 - In the API resources section, **API proxies** list, click **Add a proxy** and select the **Verification-API-v1** API proxy. Click **Add(1)**.

The settings should look as follows:

The screenshot shows the Apigee API Product configuration interface. On the left is a sidebar with icons for back, forward, search, and other management functions. The main area is titled 'API Products > ourBank API'.

- Product details:**

Name	ourBank API	Access	Public
Display Name	ourBank API	Request approval	Automatic
Description	No Description	Quota	No Quota
Environment	test	Allowed OAuth scope	No Allowed OAuth scope
- API resources:**
 - API proxies:** Verification-API-v1. A note says "This API Product will allow access to the following paths" with a box containing "/verification-api-v1/".
 - Paths:** No paths added.
- Apigee remote service targets:** Set up an Apigee remote service in order to add it to an API product.
- Custom attributes:** Key-value pairs used to store and retrieve values at runtime to control API proxy execution and customize analytics reports. No custom attributes added.

3. Click **Save** to create the new API product.

4. Navigate to **Publish > Apps** from the left menu. Then Click **+ App**.

5. Enter the following:

- **Name:** e.g. **ourBank App**
- **Display Name:** e.g. **ourBank App**
- **Company / Developer:** Developer
- **Developer:** Use the helloworld@apigee.com developer account (which has been pre-populated for the Apigee trial environment).

6. Within the Credentials section, click the **Add product** button.

7. Select **ourBank API** from the list and click **Add (1)**.

8. Review that the configuration looks as below, and click **Create**.

The screenshot shows the Apigee interface for managing APIs. On the left, there's a sidebar with icons for Home, Apps, API, Product, and Help. The main area is titled 'Apps > ourBank App'. The page is divided into three sections: 'App Details', 'Credentials', and 'Custom Attributes'.

App Details:

Name	ourBank App	App status	Approved
Display Name	ourBank App	Callback URL	
Registered	Sep 12 2020 1:49:21 AM (UTC)	Notes	
Developer	helloworld dev (helloworld@apigee.com)		

Credentials:

Status	Approved	Product	Status
Key	Show ······	ourBank API	Approved
Secret	Show ······		
Issued	Sep 12 2020 1:49:21 AM (UTC)		
Expiry	Never		

Custom Attributes:

No custom attributes.

- Once the app is created, the page displays details about the app credentials. Click **Show** on the Key row then copy the API key.

Testing

Replace `<APP_API_KEY>` with the key from the app credentials. Then, run the following curl statement in the Cloud Shell:

```
export APIKEY=<APP_API_KEY>

curl -X POST \
  https://${APIEE_ORG}-test.apigee.net/verification-api-v1/verifyCard?
apikey=${APIKEY} \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: 89236919-eabe-4357-e4c4-079f20ecd798' \
  -d '{
    "number": "2221005276762844",
```

```
"cvv": "345",
"expiration": "10/2025"
}'
```

Successful response message:

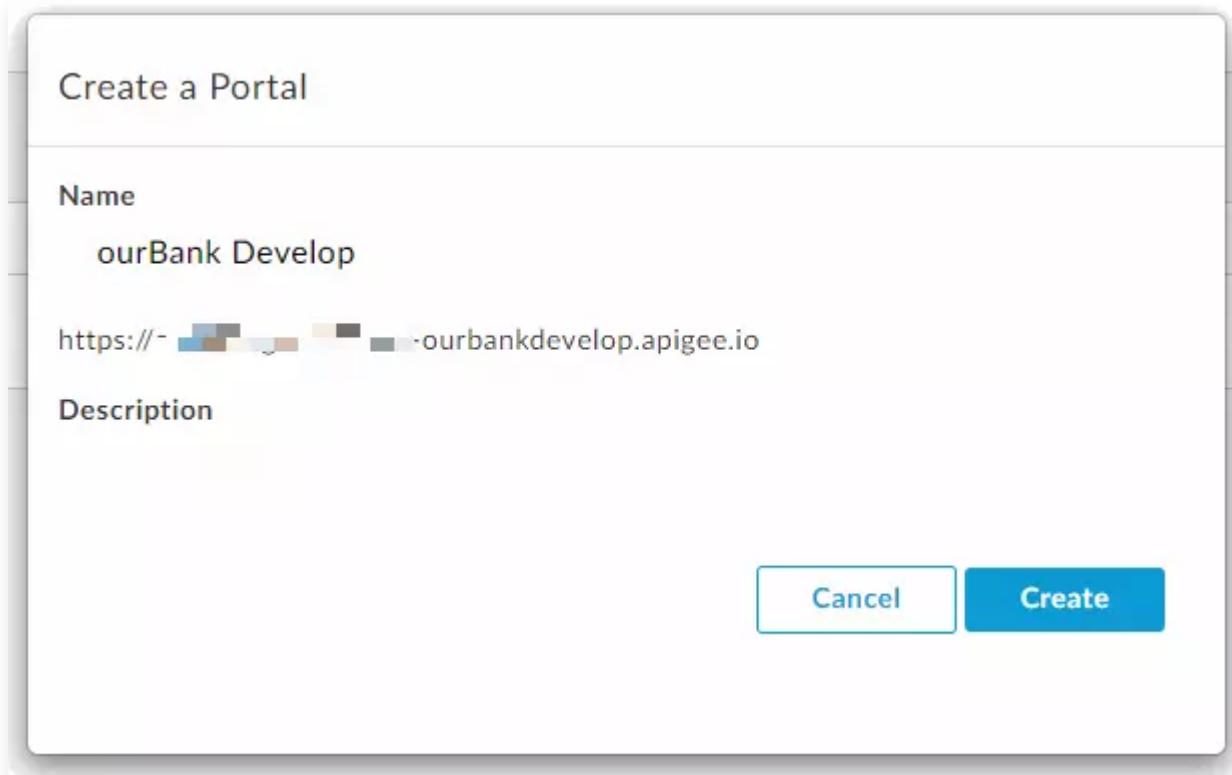
```
{"valid":true, "message":"mock response"}
```

Error response message:

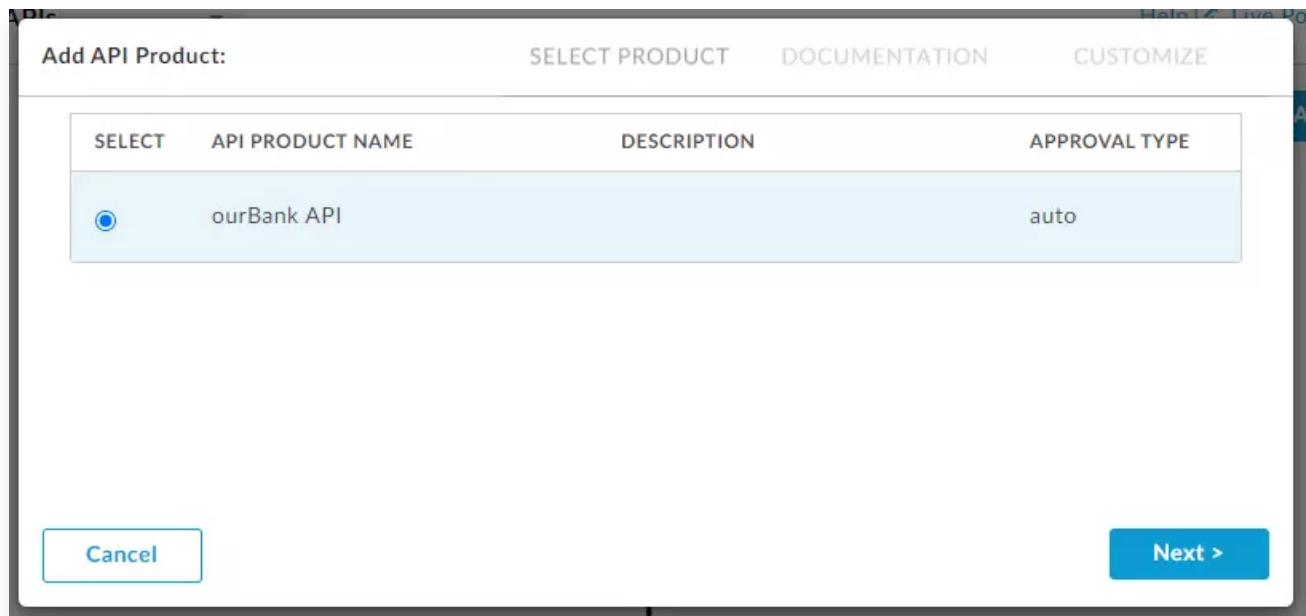
```
{"fault":{"faultstring":"Invalid ApiKey", "detail":
{"errorcode": "oauth.v2.InvalidApiKey"}}}
```

Task 3 - Create a Developer Portal for consuming the APIs

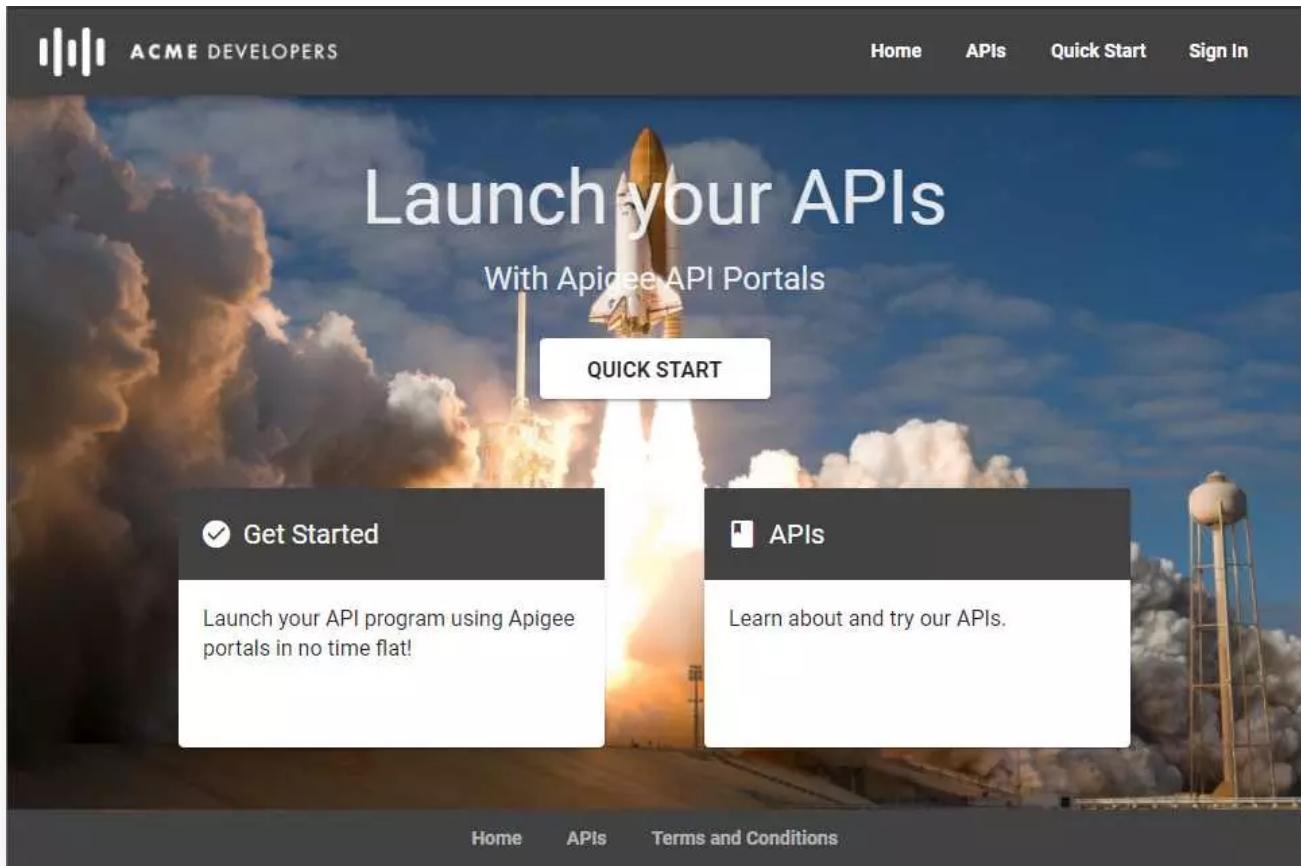
1. In Apigee, navigate to **Publish > Portals** from the left menu. Then Click **+ Portal**.
2. Enter a name, e.g. **ourBank Develop**



3. Select **APIs** from the dropdown menu at the top.
4. In the “Publish an API product” page, click on **Get started**.
5. Select the API Product (*ourBank API*) and click **Next**.



6. Click **Next >** **Next >** **Finish**.
7. Click **Live Portal** at the top-right corner to test the new developer portal.



Task 4 - Route traffic from mock response to real backend

Deploy the real backend on Cloud Function

Go the Cloud Shell, run the commands given in the lab.

```
# Make subdirectories for the code
mkdir bank-verification-service; cd bank-verification-service

# Download the code
wget https://storage.googleapis.com/apigee-quest/code/index.js
wget https://storage.googleapis.com/apigee-quest/code/package.json

# Deploy card verification function
gcloud functions deploy verifyCard --runtime nodejs10 --trigger-http --
allow-unauthenticated
```

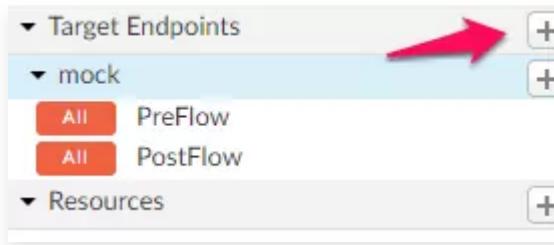
```
# Deploy address verification function
gcloud functions deploy verifyAddress --runtime nodejs10 --trigger-http --
allow-unauthenticated
```

Route traffic to these backends based on the incoming requests

1. Go back to Apigee, navigate to **API Proxies > Verification-API-v1**, click the **Develop** tab in the top right.
2. To rename the mock API endpoint, click on **Target Endpoints → default** in the Navigator tab. Replace `<TargetEndpoint name="default">` to .

```
Code default
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <TargetEndpoint name="mock">
3    <PreFlow name="PreFlow">
4      <Request/>
5      <Response>
6        <Step>
7          <Name>add-cors</Name>
8        </Step>
9      </Response>
10     </PreFlow>
11   <Flows/>
12   <PostFlow name="PostFlow">
13     <Request/>
14     <Response/>
15   </PostFlow>
16   <HTTPTargetConnection>
17     <URL>https://mocktarget.apigee.net/json</URL>
18   </HTTPTargetConnection>
19 </TargetEndpoint>
```

3. To add a new target endpoint, click on the plus sign (+) next to **Target Endpoints** in the Navigator tab.



4. Enter the following:

- **Target Endpoint Name:** e.g. `cloud`
- **HTTP Target:** _The Cloud Function Endpoint (e.g. `https://-.cloudfunctions.net_`)

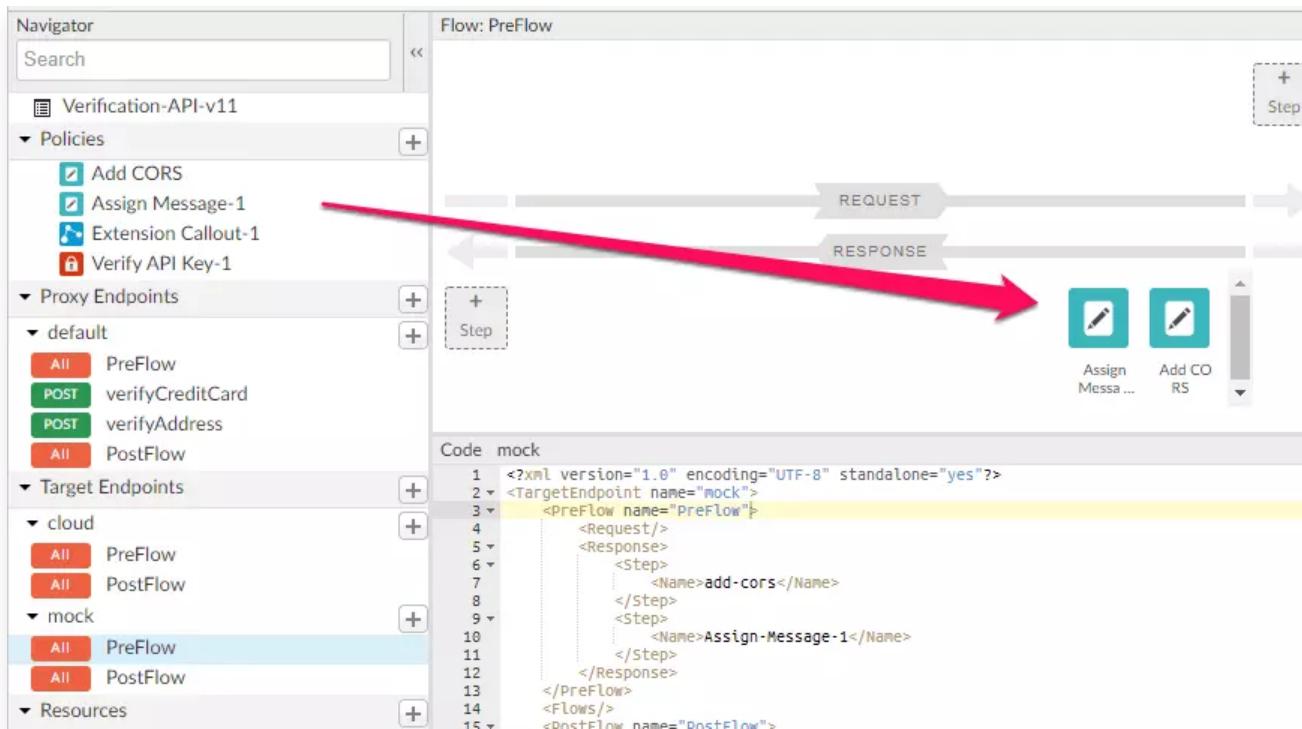
New Target Endpoint

Target Endpoint Name:

HTTP Proxy Chaining Path Chaining NodeJS

HTTP Target:

5. Click on **Proxy Endpoints** → **default** in the Navigator tab. Remove the **Assign Message-1** from the Response pipeline.
6. Click on **Target Endpoints** → **mock** → **PreFlow** in the Navigator tab. Remove the **Assign Message-1** from the Response pipeline. Drag the **Assign Message-1** from **Policies** in the Navigator tab to the Response pipeline.



7. Click on **Target Endpoints** → **cloud** → **PreFlow** in the Navigator tab. Drag the **Add CORS** from **Policies** in the Navigator tab to the Response pipeline.

8. Click on **Proxy Endpoints** → **default** → **PreFlow** in the Navigator tab. Replace the **RouteRule** tag with the following code,

```
<RouteRule name="mock">
  <Condition>request.queryparam.mock = "true"</Condition>
  <TargetEndpoint>mock</TargetEndpoint>
</RouteRule>
<RouteRule name="default">
  <TargetEndpoint>cloud</TargetEndpoint>
</RouteRule>
```

The screenshot shows the XML code editor with the previous RouteRule code highlighted with a blue selection bar. The code is part of a larger configuration block for a 'ProxyEndpoint'.

9. Click **Save**.

Testing

Test the real backend routing using the `curl` statement below.

```
curl -X POST \
  https://${APIEE_ORG}-test.apigee.net/verification-api-v1/verifyCard?
apikey=${APIKEY} \
-H 'cache-control: no-cache' \
-H 'content-type: application/json' \
-H 'postman-token: 89236919-eabe-4357-e4c4-079f20ecd798' \
-d '{
  "number": "2221005276762844",
  "cvv": "345",
  "expiration": "10/2025"
}'
```

Expected output:

```
{"valid":true, "message":"credit card is valid"}
```

Test the mock response routing using the `curl` statement below.

```
curl -X POST \
  'https://${APIEE_ORG}-test.apigee.net/verification-api-v1/verifyCard?
mock=true&apikey=${APIKEY}' \
-H 'cache-control: no-cache' \
-H 'content-type: application/json' \
-H 'postman-token: 89236919-eabe-4357-e4c4-079f20ecd798' \
-d '{
  "number": "2221005276762844",
  "cvv": "345",
  "expiration": "10/2025"
}'
```

Expected output:

```
{"valid":true, "message":"mock response"}
```