Apigee Lab 8a: Calling Services in Parallel Using JavaScript

experiment Lab schedule 1 hour 30 minutes universal_currency_alt No cost

show_chart Introductory

Overview

In this lab, you use JavaScript policies to call services in parallel.

Objectives

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

- Call APIs from JavaScript.
- Separate requests from responses in order to call APIs in parallel.

Setup and requirements

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.

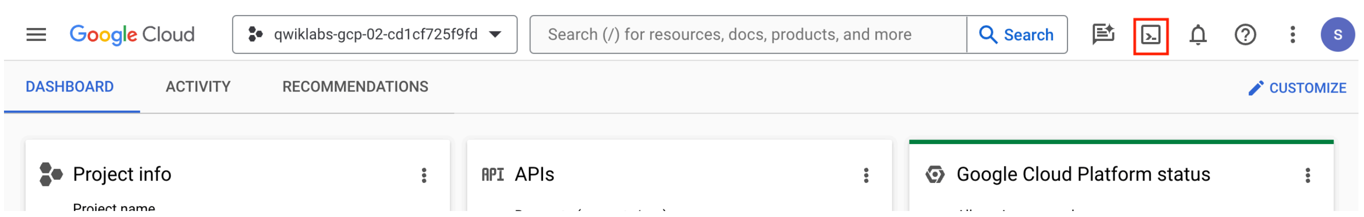
Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

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.

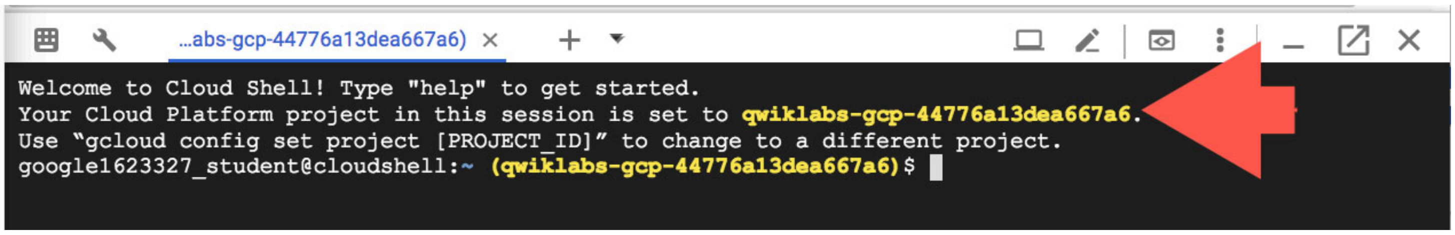
Google Cloud Shell provides command-line access to your Google Cloud resources.

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
DASHBOARD
ACTIVITY
RECOMMENDATIONS
CUSTOMIZE
Project info
Project name
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 .

Task 1. Create a new proxy

In this task, you create a new API proxy.

1. In the Google Cloud console, on the **Navigation menu** (☰), select **Integration Services > Apigee > Proxy Development > API proxies**.

2. To start the proxy wizard, click **+Create**.

3. For **Proxy template**, select **No target**.

4. Specify the following settings:

Property	Value
Proxy Name	**lab8a-v1**
Base path	**/lab8a/v1**

Note: Confirm that you are using "/lab8a/v1" for the base path, and not "/lab8a-v1".

5. Click **Create**.

6. Click the **Develop** tab.

Task 2. Add a JavaScript policy to send the requests

In this task, you create a JavaScript policy to call the Google Books API.

You will add the ability to call the book service multiple times in parallel by providing a comma-separated list of topics:

```
GET /lab8b/v1?search=java,sql,python
```

The JavaScript policy would then call the following Books API calls in parallel:

```
https://www.googleapis.com/books/v1/volumes?country=US&q=subject:java
https://www.googleapis.com/books/v1/volumes?country=US&q=subject:sql
https://www.googleapis.com/books/v1/volumes?country=US&q=subject:python
```

A second JavaScript policy will accept the responses and combine the results.

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

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

3. In the **Add policy step** pane, select **Create new policy**, and then select **Extension > Javascript**.

4. Specify the following values:

Property	**Value**
Name	**JS-SendRequests**
Display name	**JS-SendRequests**

5. For **Javascript file**, select **Create New Resource**.

6. For **Resource name**, specify `sendRequests.js`.

7. Click **Add**.

8. For **Javascript file**, select **sendRequests.js**.

9. Click **Add**.

10. Click **Resources** > **jsc** > **sendRequests.js**.

11. Paste the following code into the .js file:

content_c

```
// search is a query parameter with a comma-separated list of
topics to be searched
var searchParam =
context.getVariable("request.queryparam.search");
if (searchParam === null || searchParam.length === 0) {
  throw("search query parameter not found");
}
var search = searchParam.split(",");
print("search=" + search);

// call max 5 in parallel
var maxCalls = search.length;
if (maxCalls > 5) {
  maxCalls = 5;
}
print("maxCalls=" + maxCalls);

// URL to the Google Books API
var url="https://www.googleapis.com/books/v1/volumes?
country=US&q=subject:";

var searchTerms = [];

// for each search term, call the Google API
for (var i=0; i < maxCalls; i++) {
  var searchTerm = search[i];
  print("" + i + ": " + searchTerm);
  print("GET " + url + ""+searchTerm);
  var req = httpClient.get(url+searchTerm);

  // store the request in a session for later retrieval
  searchTerms.push(searchTerm);
  context.session["searchTerm:" + searchTerm] = req;
}

context.session["searchTerms"] = searchTerms;

print("done sending the requests");
```

This code makes a separate request for each search term passed in, up to a maximum of 5. The request is made without waiting for the response, which causes the requests to be executed in parallel.

Note: If you build a proxy like this for production use, validate the search query parameter and return an error before calling the services.

The request objects are stored in the **context.session** object. Other JavaScript policies can retrieve entities stored in context.session.

Task 3. Add a JavaScript policy to process the responses

In this task, you create a JavaScript policy to retrieve the responses and combine them into a single response.

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

   Note: You could put the JS-ProcessResponses policy in the response flow if your proxy is also going to call a backend service.

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

3. In the **Add policy step** pane, select **Create new policy**, and then select **Extension > Javascript**.

4. Specify the following values:

Property	**Value**
Name	**JS-ProcessResponses**
Display name	**JS-ProcessResponses**

5. For **Javascript file**, select **Create New Resource**.
6. For **Resource name**, specify `processResponses.js`.
7. Click **Add**.
8. For **Javascript file**, select **processResponses.js**.
9. Click **Add**.
10. Click **Policies** > **JS-ProcessResponses**.
11. Click **Resources** > **jsc** > **processResponses.js**.
12. Change the configuration for the **JS-ProcessResponses** policy:

```
<Javascript continueOnError="true" enabled="true"
timeLimit="5000" name="JS-ProcessResponses">
  <ResourceURL>jsc://processResponses.js</ResourceURL>
</Javascript>
```

content_c

You made two changes to the root element:

- The **maximum time limit** for the JavaScript policy is changed from 200 milliseconds to **5 seconds**.
- **continueOnError** is set to true, so that the call will return whatever has been successfully retrieved if the policy times out.

Note: Evaluation orgs have a documented time limit of 200 milliseconds, so the 5-second limit might not be applied. In a paid org, the timeLimit attribute should be in effect.

13. Click **Resources** > **jsc** > **processResponses.js**.

14. Paste the following code into the .js file:

content_c

```
var searchTerms = context.session["searchTerms"];
var resp = [];

// Iterate for each search term
for (var i=0; i < searchTerms.length; i++) {
  // retrieve request object from session
  var searchTerm = searchTerms[i];
  req = context.session["searchTerm:" + searchTerm];
  req.waitForComplete();
  print("Got response for " + searchTerm);

  if (req.isSuccess()) {
    print("Success!");
    item = {
      query : searchTerm,
      result : JSON.parse(req.getResponse().content)
      };
  }
  else {
    print("Error: " + JSON.stringify(req.getError()));
    item = {
      query : searchTerm,
      result : null
    }
  }
  resp.push(item);
  print(item);
  // store JSONresponse in a temporary flow variable
  context.setVariable("bookResponses", JSON.stringify(resp));
}

print("done processing responses");
```

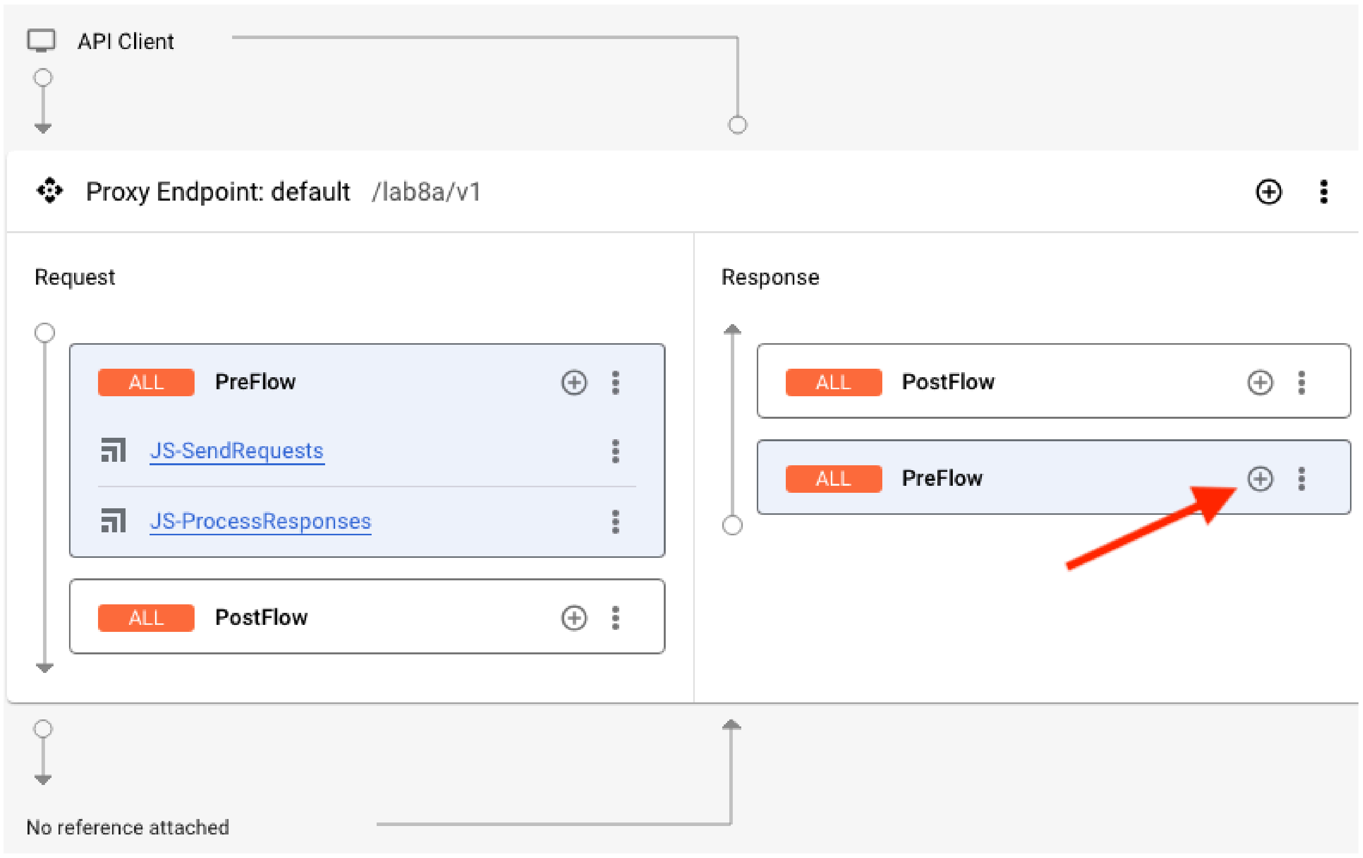
This code retrieves all of the requests previously stored in the session and waits for each to complete. All responses are combined into a single object.

Task 4. Add an AssignMessage policy to build the response

In this task, you create an AssignMessage policy to build the API response.

1. Click **Proxy 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-BuildResponse**
Display name	**AM-BuildResponse**

5. Click **Add**.

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

7. Change the policy configuration to:

```
<AssignMessage continueOnError="false" enabled="true" name="AM-
BuildResponse">
  <Set>
    <Payload contentType="application/json">{bookResponses}
</Payload>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

content_c

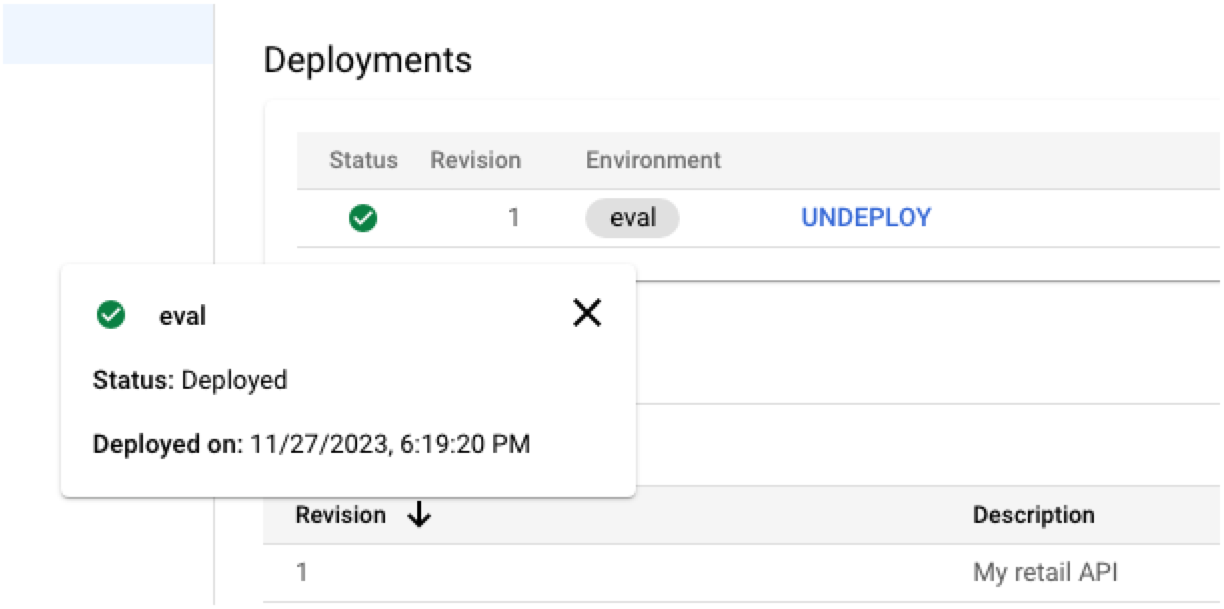
8. Click **Save**, and then click **Deploy**.

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

10. Click **Confirm**.

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

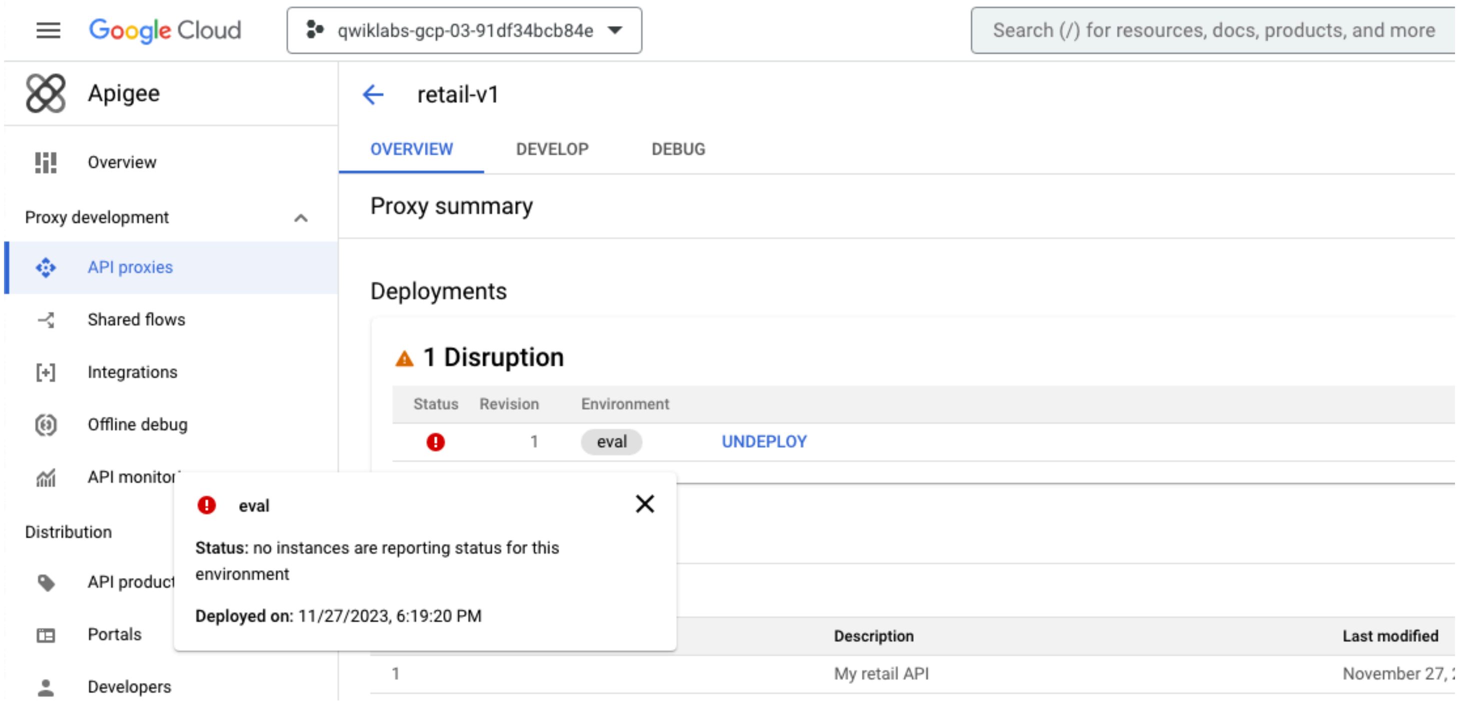
eval

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

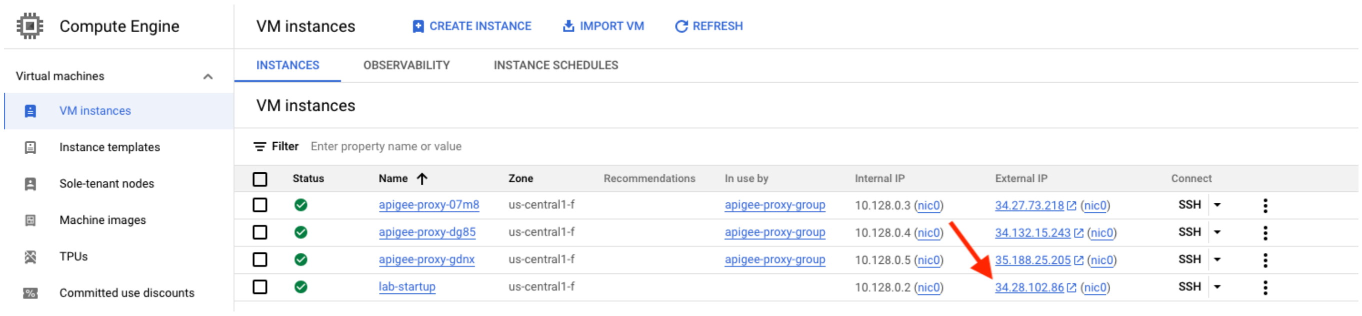
	Description	Last modified
1	My retail API	November 27,

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.

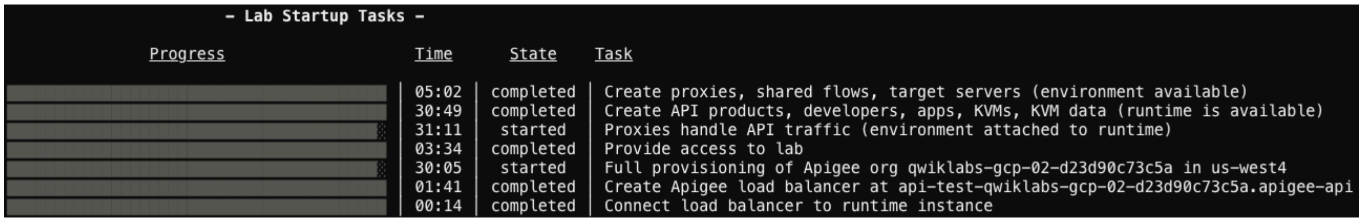


Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	apigee-proxy-07m8	us-central1-f		apigee-proxy-group	10.128.0.3 (nic0)	34.27.73.218 (nic0)	SSH
✓	apigee-proxy-dg85	us-central1-f		apigee-proxy-group	10.128.0.4 (nic0)	34.132.15.243 (nic0)	SSH
✓	apigee-proxy-gdnx	us-central1-f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

Learn more while you wait for the deployment to complete:

- JavaScript policy reference — documentation for the JavaScript policy
- AssignMessage policy reference — documentation for the AssignMessage policy
- JavaScript object model: httpClient — how to use the httpClient in a JavaScript policy to call an external service

Task 5. Debug the proxy

In this task, you use the debug tool to test and examine the proxy.

Start a debug session

1. Click the **Debug** tab, and then click **Start Debug Session**.

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

3. Click **Start**.

4. In Cloud Shell, send the following curl command:

```
curl -X GET "https://api-
test-${GOOGLE_CLOUD_PROJECT}.apiservices.dev/lab8a/v1?
search=java,sql,python" | json_pp
```

content_c

   Your response should be an array of 3 query objects, each with the search term and a result object containing the response payload. The beginning of your response should look something like this:

```
[
    {
        "query" : "java",
        "result" : {
            "items" : [
                {
                "accessInfo" : {
                    "accessViewStatus" : "SAMPLE",
                    "country" : "US",
                    "embeddable" : true,
                    "epub" : {
                        "isAvailable" : false
                    },
                    "pdf" : {
                        "isAvailable" : true
            },
            "publicDomain" : false,
            "quoteSharingAllowed" : false,
            "textToSpeechPermission" : "ALLOWED",
            "viewability" : "PARTIAL",
            "webReaderLink" : "http://play.google.com/books/reader?
        },
        "etag" : "I/aKsW0+3W4",
        "id" : "nzhxR1spWEYC",
        "kind" : "books#volume",
        "saleInfo" : {
            "country" : "US",
            "isEbook" : false,
            "saleability" : "NOT_FOR_SALE"
        },
        "selfLink" : "https://www.googleapis.com/books/v1/volumes/n
```

Note: The ordering of the queries in the response may be unpredictable because the subjects were searched in parallel.

Congratulations!

In this lab, you used JavaScript policies to call services in parallel and combine the responses into a single response.

End your lab