Apigee Edge - Pivotal Platform

Solution Paper

Ankur Shukla
December 2019

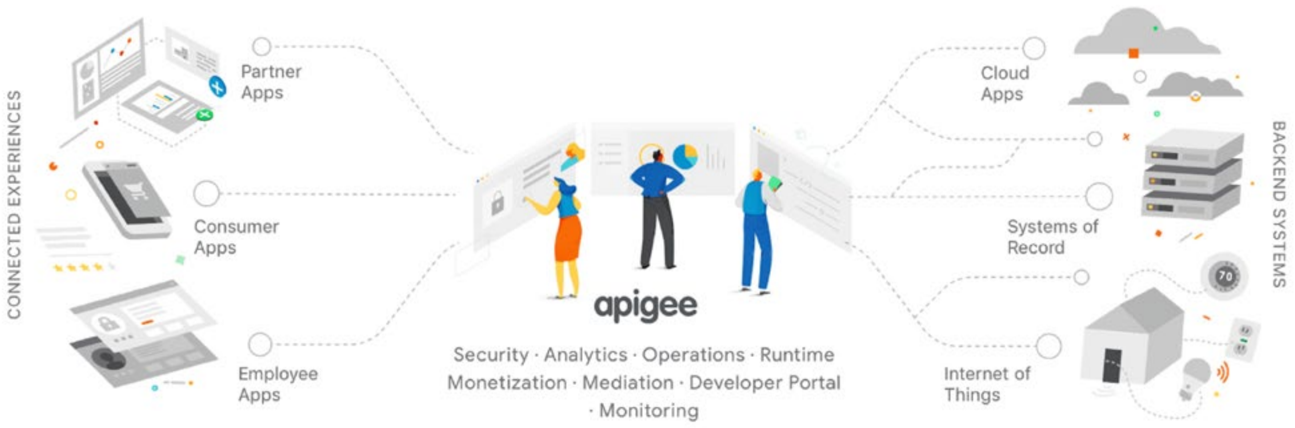
Google Cloud

This paper describes a few approaches on enabling API Management (using Apigee Edge) for your apps on Pivotal Application Service (PAS). The steps captured in this document have been validated for version 2.7 of PAS as of the publication date of this paper. You may have to make modifications as necessary for future versions.

Definitions

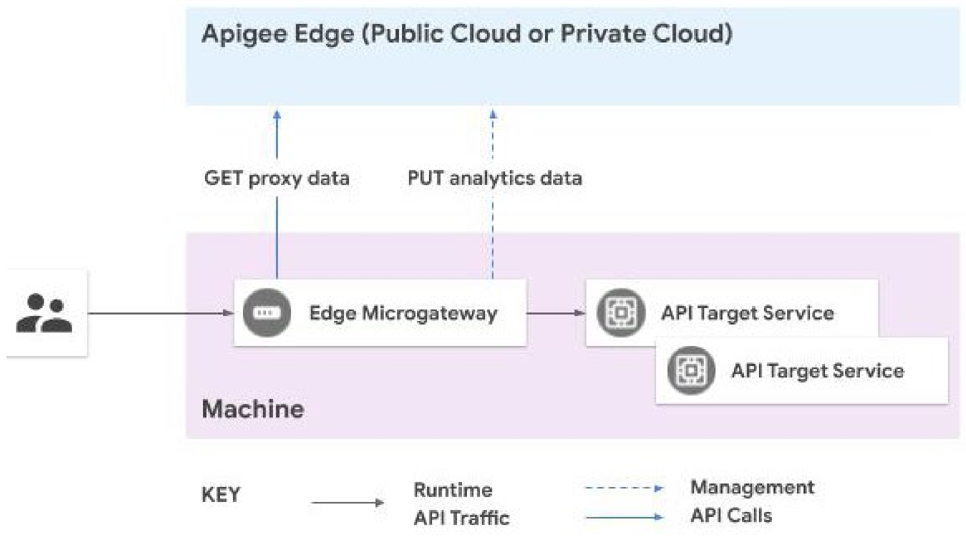
Apigee Edge

Apigee is a full lifecycle API management platform that enables API providers to design, secure, deploy, monitor, and scale APIs. Apigee sits in-line with runtime API traffic and enforces a set of out-of-the-box API policies, including key validation, quota management, transformation, authorization, and access control. API providers use the customizable developer portal to enable developers to consume APIs easily and securely as well as measure API performance and usage.



Apigee Microgateway

Apigee Microgateway is a lightweight, secure, HTTP-based message processor designed especially for microservices. Its main job is to process requests and responses to and from backend services securely while asynchronously pushing valuable API execution data to Apigee Edge, where it's consumed by the Edge analytics system.



Apigee Microgateway depends on and interacts with Apigee Edge. Apigee Microgateway must communicate with an Apigee Edge organization to function. This Apigee Edge organization instance could be running on cloud as a managed service provided by Google/ Apigee, within your data center on premises, or on your own private/ public cloud.

Apigee API Proxy

You expose APIs in Apigee Edge by implementing API proxies. API proxies decouple the app-facing API from your backend services, shielding those apps from backend code changes. An API proxy is essentially a message flow that is comprised of policies (XML configs) that execute in sequence when the API is invoked. These policies enable you to control the behavior of the underlying APIs.

What are Route Services and how do they work?

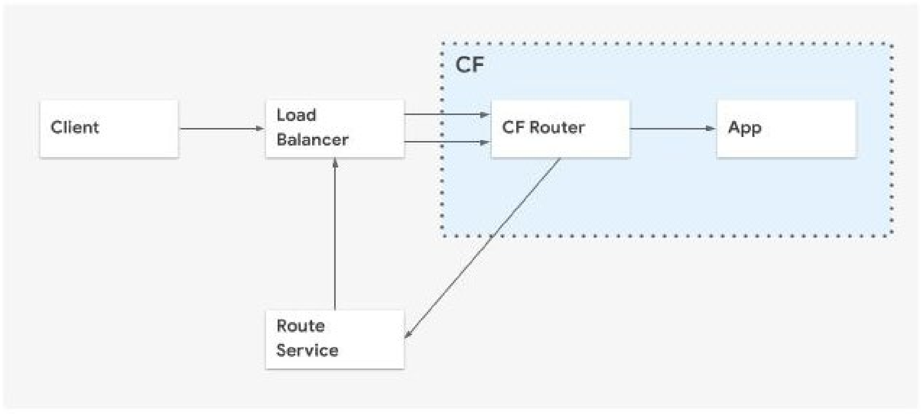
Cloud Foundry (CF) app developers may wish to apply security, transformation, or processing to requests before they reach an app. Common examples include authentication and rate limiting. Route

Services are a kind of Marketplace Service that developers can use to apply various transformations to application requests. This is typically done by binding an app's route to a service instance. Through integrations with services, providers can offer these services to developers with an automated, self-service and on-demand user experience. In its current offerings, Cloud Foundry supports the following models for route-based services:

1. Fully brokered service
2. Static brokered service
3. User-provided service

Fully brokered and static brokered require a service broker, which is typically deployed through Ops Manager as a Tile within your Cloud Foundry foundation. A user-provided service does not require a service broker and hence can be set up and configured completely by your developer. We will be using the user-provided service model as reference architecture pattern to demonstrate Apigee integrations.

The following diagram illustrates the traffic flow for a typical route-based integration using the user-provided service discussed later in this document.

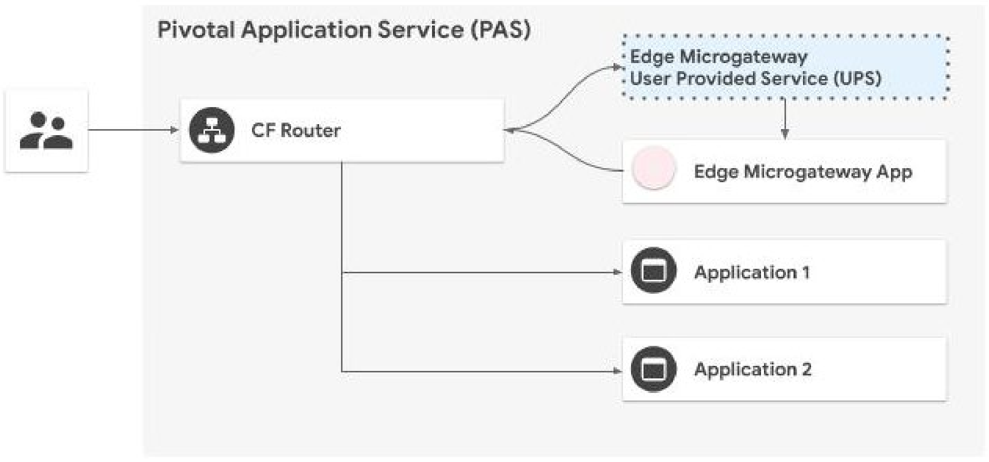


Patterns

This section describes a few patterns/approaches that can provide API Management for applications deployed on PAS. The patterns differ depending on how the traffic flows and what components of Apigee (Edge vs Edge Microgateway) end up servicing traffic.

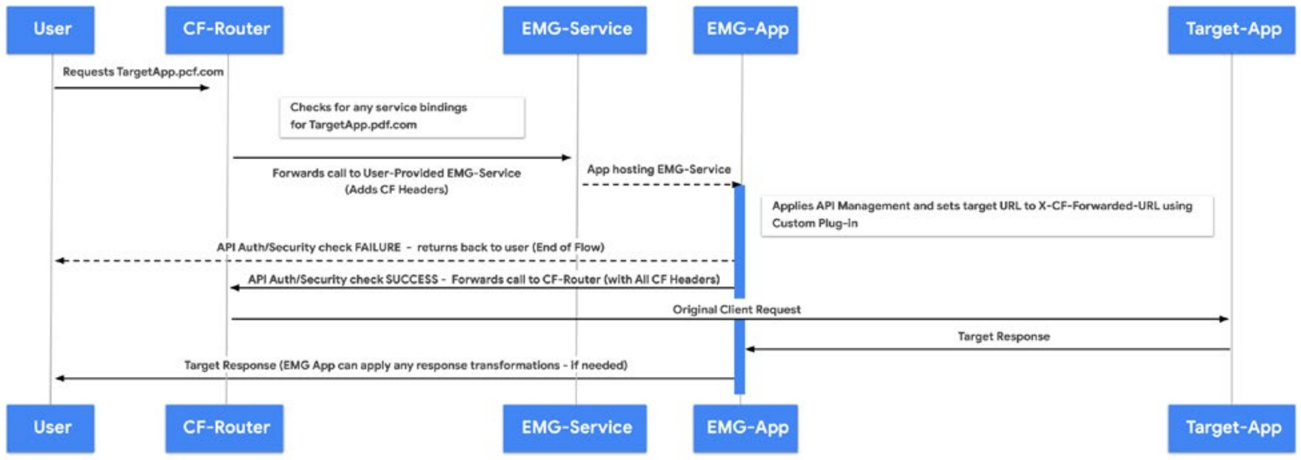
Pattern: Apigee Edge Microgateway Service on PAS

Apigee Edge Microgateway (EMG) can be hosted within the PAS platform as an application. This application can then be used to instantiate and provide a user-provided service. The user-provided service is then available for 'binding' to your applications and provides services like API security using an API key or OAuth 2.0, support for CORS, rate limiting, and so on. This integration relies upon the Route Services-based integration pattern that is offered natively by the PAS platform.


Pivotal Application Service (PAS)
Edge Microgateway User Provided Service (UPS)
CF Router
Edge Microgateway App
Application 1
Application 2

The following sequence diagram illustrates the typical flow of traffic within PAS:

Apigee Edge Microgateway (EMG) Plan using User Provided Service (Route Services Integration)


User
CF-Router
EMG-Service
EMG-App
Target-App
Requests TargetApp.pcf.com
Checks for any service bindings for TargetApp.pdf.com
Forwards call to User-Provided EMG-Service (Adds CF Headers)
App hosting EMG-Service
Applies API Management and sets target URL to X-CF-Forwarded-URL using Custom Plug-in
API Auth/Security check FAILURE - returns back to user (End of Flow)
API Auth/Security check SUCCESS - Forwards call to CF-Router (with All CF Headers)
Original Client Request
Target Response
Target Response (EMG App can apply any response transformations - if needed)
User
CF-Router
EMG-Service
EMG-App
Target-App

Setting up Microgateway as a User-Provided Service (UPS)

To use EMG as a user-provided service, we push EMG as an application to our CF environment. Once EMG is up and running as a CF application, we do the following:

- Create an EMG-aware proxy within Apigee Edge to handle CF traffic.

- Create a user-provided service by appending the CF application's (EMG) URL to the EMG proxy's base path.

Setting up an EMG application in CF, creating an EMG proxy definition in Edge, and creating a UPS is only needed to be done once. After you create the UPS, you will only need to bind each (target) application that intends to use this service.

Prerequisite
Please follow the Microgateway Operation and Configuration documentation to ensure that you have a local copy of EMG set up to communicate with your Apigee Edge organization. The "config.yaml" file, key, and secret you'll create are required to push your EMG to CF.

Create a User-Provided Service

a. Push EMG as an Application to CF:

i. In a terminal, Git clone the EMG code repo and "cd" to the "microgateway" directory.

```
git clone https://github.com/apigee-internal/microgateway.git
cd microgateway
```

ii. Edit the manifest.yml File and provide appropriate values for the following environment variables:

EDGEMICRO_KEY,
EDGEMICRO_SECRET,
EDGEMICRO_ENV,
EDGEMICRO_ORG
EDGEMICRO_CONFIG_DIR - hardcode this to value to './config'
Note - To obtain key/secret, see the Microgateway Operation and Configuration documentation.

iii. Copy your micrococogateway's config file from your home directory that was created during setup (see prerequisites), and open the new file in an editor.

```
<command> cp ~/.edgemicro/{Your-Org}-{environment}-config.yaml ./config
<command> vi ./config/{Your-Org}-{environment}-config.yaml
```

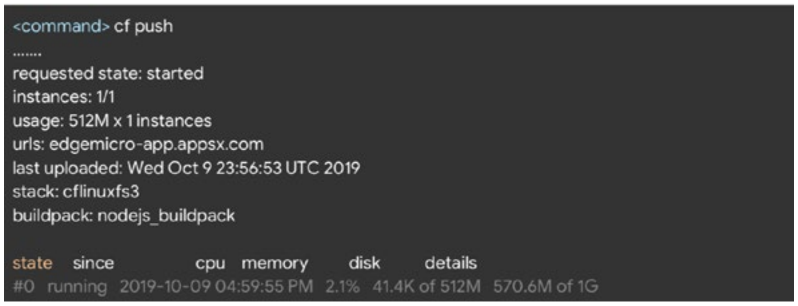
iv. Optimize the config file created in the previous step for CF by changing the port to 8080 and adding plugins as shown in the following example: (Note - The OAuth plugin is disabled to limit the scope of this solution paper).

```
edgemicro:
  port: 8080
  max_connections: 1000
  config_change_poll_interval: 600
  logging:
    level: error
    dir: /var/tmp
    stats_log_interval: 60
    rotate_interval: 24
    stack_trace: false
  plugins:
    sequence:
      - cloud-foundry-route-service
      - cloud-foundry-emg-service
      - cors
      #- oauth
      - spikearrest
cors:
  cors-origin: '*'
spikearrest:
  timeUnit: minute
  allow: 10
```

- **v.** Include the custom plugin cloud-foundry-emg-service within the plugin sequence section (see previous example). This plugin handles the routing aspect of the calls to ensure that the handshake with the CF Router happens seamlessly. The sample code for this plugin is available here. (This is likely to be changed). Create a plugin directory and include the files (index.js & package.json) from the GitHub repo. For more information on how to use plugins with EMG, see Use plugins in the Apigee documentation.
- **vi.** In the Apigee EdgeUI, login to your Apigee organization and create an EMG-aware proxy with the base path set to net'. (Later, the 'cloud-foundry-emg-service' plugin will overwrite this URL with the target CF application's URL). For reference on how to create an EMG-aware proxy, see Setting up and configuring Edge Microgateway in the Apigee documentation.

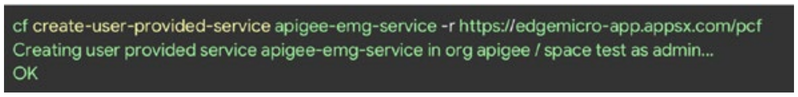
Note - Since we have not enabled the OAuth plugin within our EMG service, we do not need to follow the rest of the above documentation that details the process for creating API products, a developer, and a developer application. However, If you would like to use EMG's OAuth capabilities, you will need to create a product, developer, and application.

- **vii.** Your EMG application is now ready to be pushed to Cloud Foundry:



```
<command> cf push
.......
requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: edgemicro-app.appsx.com
last uploaded: Wed Oct 9 23:56:53 UTC 2019
stack: cflinuxfs3
buildpack: nodejs_buildpack

state    since                   cpu    memory       disk         details
#0  running  2019-10-09 04:59:55 PM  2.1%  41.4K of 512M  570.6M of 1G
```

b. Create the UPS using the cf create-user-provided-service command, providing the URL for EMG from step vii appended with the basepath of the EMG-aware proxy from step vi) '/pcf'.

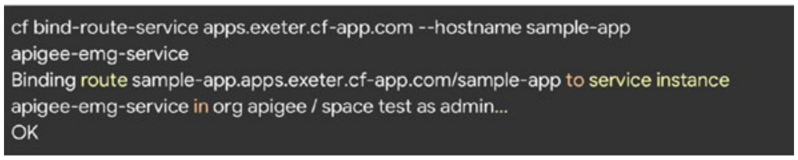


```
cf create-user-provided-service apigee-emg-service -r https://edgemicro-app.appsx.com/pcf
Creating user provided service apigee-emg-service in org apigee / space test as admin...
OK
```

This completes the UPS one-time setup steps. The service is ready to be used for binding by applications that require API management.

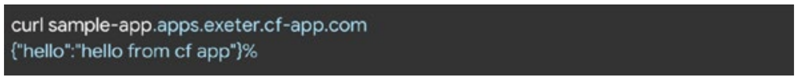
Bind a Target Application to the EMG Provided Service
Push your CF target application that requires API management capabilities from EMG and take note of App Name and Application's URL.

a. Now we are ready to bind our sample application to the UPS (EMG) using the bind-route-service command:



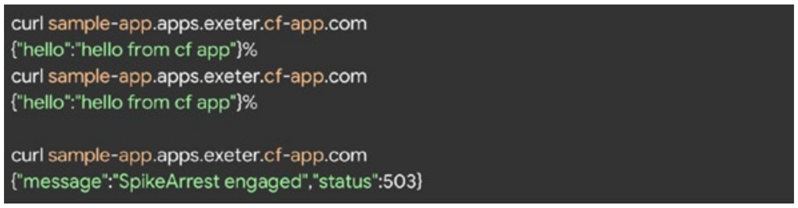
```
cf bind-route-service apps.exeter.cf-app.com --hostname sample-app
apigee-emg-service
Binding route sample-app.apps.exeter.cf-app.com/sample-app to service instance
apigee-emg-service in org apigee / space test as admin...
OK
```

b. Let's Test our binding by making cURL calls:



```
curl sample-app.apps.exeter.cf-app.com
{"hello":"hello from cf app"}%
```

Now repeat the cURL call in quick succession and you should see a spike arrest violation from EMG.



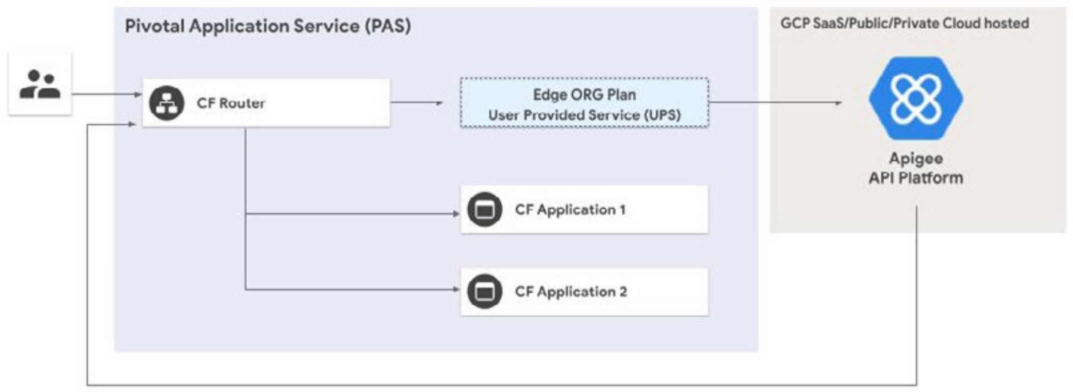
```
curl sample-app.apps.exeter.cf-app.com
{"hello":"hello from cf app"}%
curl sample-app.apps.exeter.cf-app.com
{"hello":"hello from cf app"}%

curl sample-app.apps.exeter.cf-app.com
{"message":"SpikeArrest engaged","status":503}
```

Congratulations!! You have now secured your sample application with Apigee Edge Microgateway.

Pattern: Apigee Edge Integration with PAS

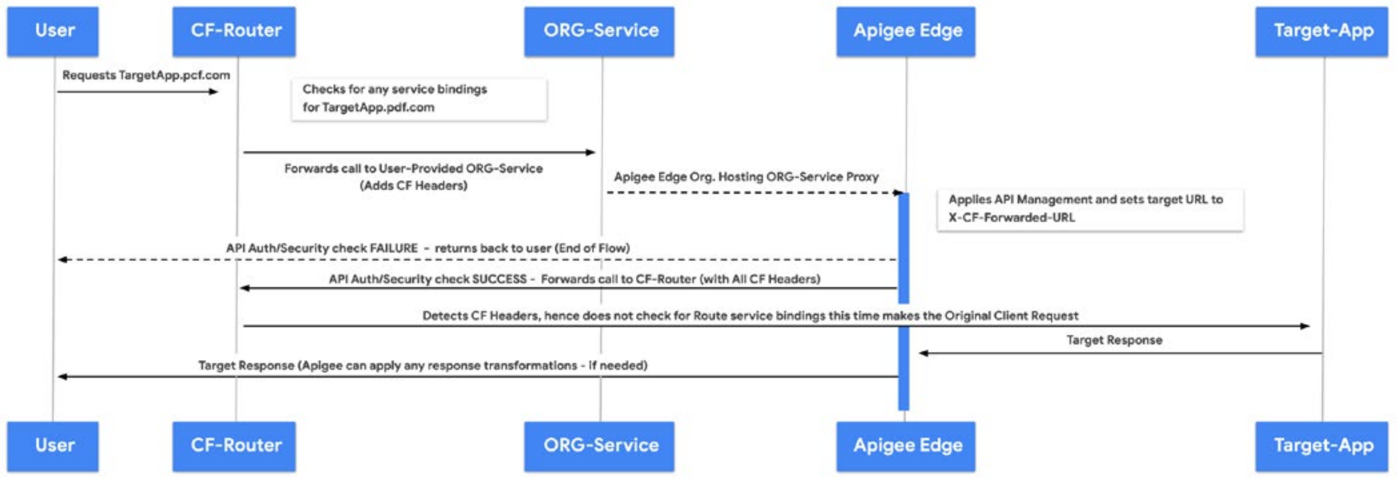
Similar to Apigee Edge Microgateway as a service (previous section), you can create a user-provided service (UPS) with your Apigee Edge organization to proxy your application hosted on PAS. The advantage of using an Apigee Edge organization as a UPS is that you can use all the capabilities of the Apigee Edge platform (45+ out of box policies) and use advanced UI features like trace, since all your traffic will flow through Apigee Edge. This Apigee Edge 'organization' can be hosted in Apigee Cloud or your own on-premises datacenter (using Apigee on Premise Deployment Kit - OPDK).

ORG Plan Overview


Pivotal Application Service (PAS)
CF Router
Edge ORG Plan
User Provided Service (UPS)
CF Application 1
CF Application 2
GCP SaaS/Public/Private Cloud hosted
Apigee
API Platform

Similar to the user-provided EMG service, the organization service is also available for 'binding' to your applications and provides services like API security using an API key or OAuth 2.0, support for CORS, rate limiting features, and more. The following sequence diagram illustrates the typical flow of traffic within PAS:

Apigee Edge (ORG) Plan using User Provided Service (Route Services Integration)


User
CF-Router
ORG-Service
Apigee Edge
Target-App
Requests TargetApp.pcf.com
Checks for any service bindings for TargetApp.pdf.com
Forwards call to User-Provided ORG-Service (Adds CF Headers)
Apigee Edge Org. Hosting ORG-Service Proxy
Applies API Management and sets target URL to X-CF-Forwarded-URL
API Auth/Security check FAILURE - returns back to user (End of Flow)
API Auth/Security check SUCCESS - Forwards call to CF-Router (with All CF Headers)
Detects CF Headers, hence does not check for Route service bindings this time makes the Original Client Request
Target Response
Target Response (Apigee can apply any response transformations - if needed)

Setting up Apigee Edge Organization as a User-Provided Service (UPS)

To use the Apigee Edge organization (org) as a user-provided service, we will create and deploy a proxy within Apigee Edge to make it available for API traffic. This proxy will handle CF traffic, and you'll use the proxy path to create a UPS within the PAS environment. Again, in this pattern too, creating a proxy in your Apigee org and UPS creation are only done once. After you configure the UPS, you will only need to bind each (target) application that intends to use this service.

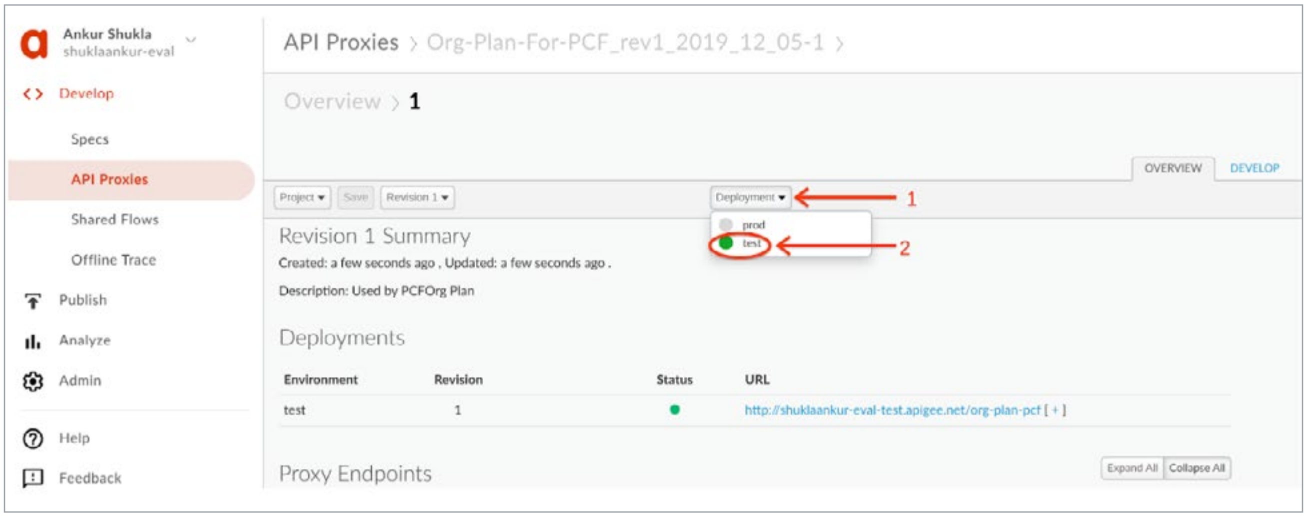
Prerequisite

Signup and register for a free Apigee Edge Evaluation Organization. See https://cloud.google.com/apigee.

Create a User-Provided Service

a. Create a proxy to act as a facade for your all applications on CF that require API management:

- **i.** Use the proxy bundle in this location to create a proxy within your Apigee Edge org by using the "Import a Proxy Bundle" dialog within the Apigee Edge UI. You can also import this proxy using the Apigee Management API.
- **ii.** Ensure that your proxy is deployed to the desired environment, illustrated by the following image. Be sure   that your virtual host in Edge uses HTTPs, as CF requires an HTTPS endpoint for the route services integrations to work.



Revision 1 Summary

Created: a few seconds ago , Updated: a few seconds ago .

Description: Used by PCFOrg Plan

Deployments

Environment	Revision	Status	URL
test	1	●	http://shuklaankur-eval-test.apigee.net/org-plan-pcf [+]

Proxy Endpoints

Note - This proxy is a passthrough proxy and does not have any security built in. However, since this proxy is running on Apigee Cloud and will be picking up traffic from CF, you are free to use and implement the OAuth 2.0 or Verify API Key policies in addition to any of the other 45+ Apigee policies. The current implementation of this proxy handles the routing of the CF call to the correct CF target application that the original call was intended for.

iii. Create the UPS using cf create-user-provided-service command (provide the URL for the Org Plan Proxy we deployed in Step ii).

```
cf create-user-provided-service apigee-emg-service -r https://edgemicro-app.appsx.com/pcf
Creating user provided service apigee-emg-service in org apigee / space test as admin...
OK
```

This completes the UPS one-time setup. The service is ready to be used for binding by applications that require API management.

Bind a Target Application to the Org UPS

b. Push your CF target application that requires API management capabilities from the Apigee org and take note of App Name and Application's URL.

i. Now you are ready to bind the sample application to the user provided service (org) using the bind-route-service command:

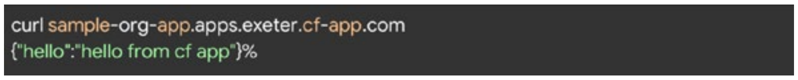
```
cf bind-route-service apps.exeter.cf-app.com --hostname sample-org-app
apigee-org-service
Binding route sample-org-app.apps.exeter.cf-app.com/sample-app to service instance
apigee-org-service in org apigee / space test as admin...
OK
```

ii. Test the binding by making cURL calls:

```
curl sample-org-app.apps.exeter.cf-app.com
{"hello":"hello from cf app"}%
```

Now Turn on the trace feature within Apigee Edge and repeat the cURL call. You should be able to see the API call traversing through Apigee Edge.

Note: If you are unfamiliar with how to use Apigee Trace tool, see Using the trace tool in the Apigee documentation.



```
curl sample-org-app.apps.exeter.cf-app.com
{"hello":"hello from cf app"}%
```

Congratulations!! You have now secured your sample application with and Apigee Edge org.

Summary - Final thoughts...

In this solution paper we discussed a couple of approaches on how Apigee could be used within PAS for API Management. In both the approaches we have relied on a single proxy creation within Apigee Edge to manage our runtime traffic for all CF Applications. This is an optimized approach as it induces less overhead by relying on one proxy / service for 'N' apps; in this scenario the analytics for the N apps, will be aggregated under the API proxy. Another approach obviously will be to create a separate proxy within Apigee Edge and then a new UPS using the proxy base paths. This will mean that you have one proxy and one UPS for each application. This approach induces extra overhead but has the side benefit of visualizing analytics of each individual app.

Resources

Microgateway custom plugin for the EMG pattern:
https://github.com/ankurshukla80/cloud-foundry-emg-service

Proxy bundle for running the Apigee org pattern:
https://github.com/ankurshukla80/cloud-foundry-emg-service/blob/master/Resources/Org-Plan-For-PCF_rev1_2019_12_05.zip