

**Appendix 5 - Microgateway**

Apigee Edge Microgateway is a secure, HTTP-based message processor for APIs. Its main job is to process requests and responses to and from backend services securely while asynchronously pushing valuable API execution data/meta data to and from Apigee Edge.

Typically, Edge Microgateway is installed within a trusted network in close proximity to backend target services; often, the μ-gateway will be co-located on the same OS image as the target service. μGwy provides enterprise-grade security, and some key plugin features such as spike arrest, quotas, and so on. The μ-gateway leverages an Apigee Edge Public Cloud instance, or an Apigee Edge private cloud (on premises) installation for other aspects of API Management like analytics, and developer management. You can install Edge Microgateway in the same data center or even on the same machine as your backend services.

### Key features and benefits

* **Traffic management:** APIs proxied through Edge Microgateway never need to leave the perimeter of your corporate network. You get the Analytics and security capabilities of Apigee Edge public cloud, without sending your transaction data over public networks.
* **Security:** Edge Microgateway authenticates requests with a signed bearer token or API key issued to each client app by Apigee Edge. This means you can secure your internal-to-internal APIs without writing new server-side code.
* **Rapid deployment:** Unlike a full deployment of Apigee Edge, you can deploy and run an instance of Edge Microgateway within a matter of minutes.
* **Network proximity:** You can install and manage Edge Microgateway in the same machine, subnet, or data center as the backend target APIs with which Edge Microgateway interacts. Same-machine proxy deployment means that proxied traffic does not incur additional network capacity, and does not require an additional VM. Super low footprint.
* **Analytics:** Edge Microgateway asynchronously delivers API execution data to Apigee Edge, where it is processed by the Edge Analytics system. You can use the full suite of Edge Analytics metrics, dashboards, and APIs.
* **Reduced latency:** All proxy activity is done on the local datacenter network, meaning the overhead of the proxy is very very low. All communication with Apigee Edge is asynchronous and does not happen as part of the client request pipeline. This allows Edge Microgateway to collect API data and send it to Apigee Edge without affecting latency of the runtime transactions.

## Objectives

In this lab you will go through configuring, get working Edge Microgateway installation capable of processing an API. You'll make several secure, test API calls through Edge Microgateway to the API's backend service and see how Apigee Edge processes analytics data from Edge Microgateway.

## Prerequisites

*   Edge Micro ZIP File is downloaded to local folder. download from [here](https://www.dropbox.com/s/02jdmvzfozugkoo/apigee-edge-micro-1.1.0.zip?dl=0)
*   Node 4.2.1 or later is installed

node -v v4.2.1

*   Have access to Edge Org
*   The Node.js service you have created in [Lesson 6](https://github.com/apigee/apigee-partner-se-bootcamp/blob/master/labs/API%20Services%20Lesson%206%20-%20Create%20a%20Node.js%20Proxy) is up and running. This lesson uses that servics as a backend

## Estimated Time: 60 mins

### Configure Edge Micro

* Unzip the file unzip apigee-edge-micro-1.1.0.zip
* Go to the bin folder cd apigee-edge-micro-1.1.0/cli/bin
* Test the installation by executing the following command. If it returns CLI help output, then the installation was successful.

./edgemicro -h

* Put the cli/bin directory in your PATH variable. This way, you'll be able to execute Edge Microgateway commands from anywhere.

export PATH=<Edge\_Micro\_Gateway\_Installaton\_Folder>/cli/bin:$PATH echo $PATH

**Note:** Throughout the tutorial, we assume that the cli/bin directory is in your PATH. If you do not put it in your PATH, then you need to execute CLI commands from the cli/bin directory. OR, you need to prepend the ./cli/bin directory to all edgemicro commands.

* Configure the micro gateway to talk to the Edge deployment on public/private cloud

./edgemicro configure -o <org-name> -e <env-name> -u <your Apigee email>

* **Copy the** Key, Secret and vault info, for later parts of the exercise.
* Using the key and secret you just viewed, verify that the edge micro is configured correctly with the Edge Cloud org.

./edgemicro verify -o <org-name> -e <env-name> -k <key> -s <secret>

You should see happy messages. If not, you'll need to stop, and diagnose.

### Start the Edge Micro agent

* Before continuing, open a separate terminal window. This is where the agent server will run.
* cd to the agent directory.  cd ../../agent
* Execute this command to start the server:  npm start  You will see an output from that command:  edge micro agent listening on 9000

The configuration done so far allows Edge Microgateway to bootstrap itself to Apigee Edge. The Edge Microgateway agent manages this bootstrapping process, which kicks off whenever you start or restart Edge Microgateway. After the bootstrapping succeeds, Edge Microgateway retrieves a payload of additional configuration information from Apigee Edge.

### Create a passthrough API Proxy

Log in to your organization on Apigee Edge.

* Select APIs > API Proxies from the top menu.
* In the API Proxies page, click + API Proxy.
* In the dialog, configure the proxy as follows:
  + Starting Point Type: **Backend Service**
  + Backend Service URL: [**https://weather.yahoo.com**](https://weather.yahoo.com/)
  + Name: edgemicro\_{your\_initials}\_weather
  + Project Base Path: /{your-initials}/v1/weather
  + Description: Edge Micro Proxy that will run locally to the node.js service
  + Features: None

**Note:** Edge μ-gateway-aware proxy names must always begin with the prefix **edgemicro\_**.

* Click on Build to build and deploy the proxy
* Once the proxy has been deployed, click on the Close button
* Update target endpoint
  + Switch to the Develop tab of the API Proxy
  + From the Navigator pane, select `Target Endpoints → Default
  + In the code view, update the HTTPTargetConnection → URL to http://localhost:7000
* Click on the Save button to save and deploy the changes to the hotels API Proxy
* We have created an API Proxy that will be fetched by Edge Micro gateway, when it starts locally
* Add the API Proxy you created edgemicro\_<your initials>\_weather to the API Product <your initials> Hospitality Basic Product
  + From the Apigee Edge Management UI, go to Publish → API Products
  + Select the API Product: <your initials> Hospitality Basic Product
  + Click on Edit Button on the top right
  + Click on + API Proxy button and API Proxy edgemicro\_<your initials>\_weather

### Start Edge Microgateway

* Open a new terminal window (not the one where the agent is running) and print out help info for the agentproc command. We're going to use the -c, -k, and -s options to start Edge Microgateway.

./edgemicro agent proc -h

**Note:** The tutorial assumes that the cli/bin directory is in your PATH. If you do not put it in your PATH, then you need to execute CLI commands from the cli/bin directory (cd ../cli/bin and run ./edgemicro agent proc -h).

* Start Edge Microgateway by executing the following command, providing the key and secret values with the -k and -s parameters, respectively.

./edgemicro agent proc -c start -k <key> -s <secret>

* If the micro gateway is started successfully, you will see something like the following console output:

{ "pid": 9104, "uid": "MTQ0NzQ1MjU4Nzk0MQ", "running": "true", "restarts": 0, "since": "2015-11-13T22: 09: 47.959Z" }

**Note:** The **UID** is the unique id for this instance of Edge Microgateway. You use this ID when retrieving log and monitoring information about Edge Microgateway. See the Edge Microgateway Administrator's Guide for more information.

**Agent output:** In the terminal where the agent is running, you'll see output like this. You may have to scroll up through the terminal output to see some of the messages.

info: config download from https://edgemicroservices-us-east-1.apigee.net/edgemicro/bootstrap/organization/amer-partner1/environment/test returned 200 OK info: products download from https://amer-partner1-test.apigee.net/edgemicro-auth/products returned 200 OK info: jwt\_public\_key download from https://amer-partner1-test.apigee.net/edgemicro-auth/publicKey returned 200 OK warning: no edge micro proxies found in org warning: no products found in org downloaded jwt\_public\_key '\*\*\*\*\*' installed plugin from /Users/SandeepM/Apigee/Products/EdgeMicro/apigee-edge-micro-1.1.0/plugins/analytics installed plugin from /Users/SandeepM/Apigee/Products/EdgeMicro/apigee-edge-micro-1.1.0/plugins/oauth MTQ0NzQ0NjgwNzkxOQ edge micro listening on port 8000

At this point, the agent retrieves a payload of Edge Microgateway configuration information from Apigee Edge. This information includes:

* +   The public key we created and stored previously in the Apigee vault.
  +   A JSON representation of all Edge Microgateway­aware proxies that exist in the organization/environment. These are all proxies that are named with the prefix edgemicro\_.
  +   A JSON representation of all of the API products that exist in the organization.

With this information, Edge Microgateway knows which proxies and proxy paths are allowed. If/when configured, it uses the product information to enforce security (in the same way as any API proxy does on Apigee Edge, where developer app keys have an association with products). We'll go through the steps to secure Edge Microgateway shortly.

* To test our API Proxy, call the

curl -i http://localhost:8000/{your-initials}/v1/weather/forecast?zipcode=98101

* If you will see an authorization error, your Micro Gateway is listening for the API you have configured in the previous step

{"error":"missing\_authorization","error\_description":"Missing Authorization header"}

* Run the following command to get the JWT Token

curl -i -X POST "http://<org name>-test.apigee.net/edgemicro-auth/token" \ -H "Content-Type:application/json" \ -d '{ "grant\_type": "client\_credentials", "client\_id":"...CLIENT\_ID\_HERE...", "client\_secret": "...CLIENT\_SECRET\_HERE..." }'

The response you see will have a very long string, looking something like so:

"eyJ0eXAiOiJKV1QiLc9.eyJhcHBsaWNhdGlvbl9u....kkBnE40jk\_2trmZkP6uf4-mcgUw91-qKofaw"

* Make the API Call with a bearer token you received in the step above

curl -i -H "Authorization:Bearer VERY\_LONG\_JWT\_TOKEN\_HERE" http://localhost:8000/{your-initials}/v1/weather/forecast?zipcode=95113

* You should see the response json:

{ "location": { "city": "San Jose", "country": "US", "region": "CA" }, "units": { "distance": "mi", "pressure": "in", "speed": "mph", "temperature": "F" }, "condition": { "code": "34", "date": "Fri, 13 Nov 2015 12:53 pm PST", "temp": "63", "text": "Fair" }, "forecast": [ . . . ] }

You have used the credentials you received from your Edge org on cloud and you have used them on your local Micro Gateway. You are able to manage your APIs centrally, while the enforcement is happening locally.

## Summary

That completes this hands-on lesson. In this lesson you learned about setting up an Edge Micro Gateway, creating an API Proxy and corresponding life cycle on your Edge org. You now have a fully functioning and secure Edge Microgateway that can run your APIs locally, picking up the security credentials from your Edge org.