Welcome to the management module of the On Premises Management, Security, and Upgrade course of Google Cloud's Apigee API Platform for private cloud.

Agenda

Platform operation

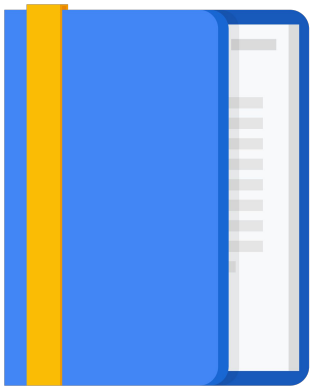
Infrastructure operation

REST fundamentals

Management API

API proxy deployment

Edge analytics



In this module we discuss various aspects of platform and infrastructure operation of the Apigee private cloud platform.

We discuss the fundamental concepts of REST and provide an overview of the Apigee management API for private cloud.

You learn about API proxy deployment and how you can use Apigee analytics to track the operational performance of the platform.

Let’s begin!

Directory structure

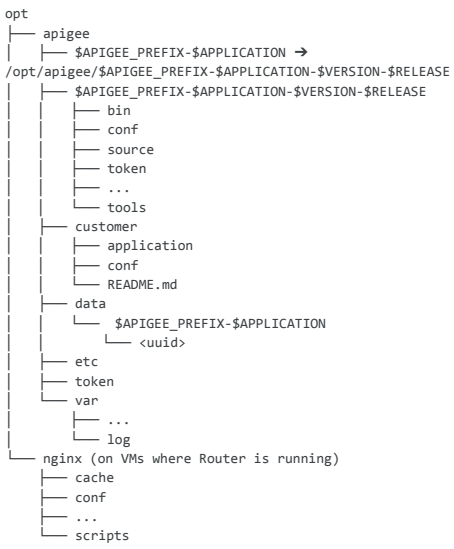
Edge installation file structure is as shown.

`$APIGEE_PREFIX-$APPLICATION`

Examples:

`apigee-cassandra`

`edge-router`



In this lesson we'll discuss several aspects of Apigee Edge management.

We will start with the directory structure and naming conventions, service control, and component status.

All Apigee components are installed in the base directory /opt/apigee with the exception of Nginx, which is installed in /opt/nginx.

If you need to deploy Apigee components on a different path, you can do so by creating symlinks from the desired directory to /opt/apigee and /opt/nginx before you start the installation.

Within /opt/apigee, there are several subdirectories. Depending on the type of component that has been installed, there will be one or more application directories.

These directories are named following the format $APIGEE_PREFIX-$APPLICATION and contain binaries, scripts and libraries for each component.

When a service is installed, the actual directory naming structure follows the pattern $APIGEE_PREFIX-$APPLICATION and also includes the release version number.

A symlink is also created to this directory.

Directory structure

Path	Purpose
/opt/apigee/token	Install-time configuration customizations
/opt/apigee/customer	Customer-specific configuration customizations
/opt/apigee/data	Service data files (example: Cassandra database files)
/opt/apigee/etc	Edge internal settings (do not edit)
/opt/apigee/var	Locks, logs, and PID files used by components at runtime
/opt/apigee/$APIGEE_PREFIX-$APPLICATION	Binaries, scripts, and libraries for each component

The other directories that are created under /opt/apigee are:

token: this directory contains install time configuration customizations specified in a config file.

customer: this directory has customer specific configuration customizations that persist across upgrades of the Apigee software.

data: this directory contains service data files (for example, cassandra database files) and also contains the UUIDs of the components generated during installation.

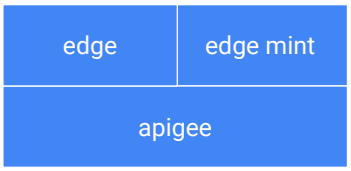
etc: this directory contains Apigee Edge internal settings, which should not be edited.

var: this directory contains logs, locks, and pid files that are used by components at runtime.

Component naming convention

Some subdirectories under `/opt/apigee` use a component-specific naming convention consisting of a prefix and the component name:

Prefix	Description	Example
apigee	An open source component used by Edge	apigee-cassandra
edge	An Edge component	edge-management-server
edge-mint	A Monetization component	edge-mint-management-server



When naming components, Apigee use three prefixes that help to identify the component type.

The prefix can be one of the following:

apigee: for an open source component used by Apigee Edge, for example, apigee-cassandra.
edge: an Edge component developed by Apigee, for example, edge-management-server.
edge-mint: an Edge monetization component, for example, edge-mint-management-server.

This naming convention is also used in the Apigee private cloud directory structure.

Service control: apigee-service

The apigee-service utility provides unified service control and component management.

- The full path to the utility is: `/opt/apigee/apigee-service/bin/apigee-service`
  - Usage takes the format: `apigee-service <component> <action> [options]`
  - This directory is also added to the system search path.
  - Use the tab key to complete apigee-service component names and actions.
- Use apigee-service to manage the runtime state and internal configuration of each component:

```
> apigee-service apigee-cassandra stop
> apigee-service edge-router restart
> apigee-service edge-management-server store_ldap_credentials
> apigee-service apigee-postgresql setup-replication-on-standby
  -f /tmp/config.txt
```

Let's now discuss service control.

Edge has a single utility called **apigee-service**, that provides unified service control and component management. The command is located in the /opt/apigee/apigee-service/bin directory.

The command takes the form: **apigee-service** followed by a **component name**, an **action** and **options**.

To make it easier to use, the apigee-service bin directory is added to the system search path at installation time.

Type completion can also be used to show available components and actions.

Service control: apigee-all

The apigee-all command applies apigee-service actions to all components found on a host.

- The command is at `/opt/apigee/apigee-service/bin/apigee-all`.
  ```
  > apigee-all stop
  > apigee-all restart
  > apigee-all status
  > apigee-all version
  > apigee-all backup
  > apigee-all wait_for_ready
  ```
- To see all available components, run: `apigee-service help`
- To see all available actions for a component, run:
  `apigee-service <component> <tab> <tab>` (tabbed completion)

The **apigee all** command can apply apigee-service actions to all the service components found on a host.

You can use the **apigee all version** command to check that the versions of all installed components match, and **wait_for_ready** to check that the components are running and all startup routines are completed.

Components status

Apigee Edge components expose a service management API on a custom port.

- Valid API calls vary by component, but all components support a set of core calls used to check status:
  - /v1/servers/self
  - /v1/servers/self/up
  - /v1/servers/self/uuid
- To call a component API, use curl or your HTTP client of choice:

```
> curl http://<router-ipaddr>:8081/v1/servers/self/up
```

- Additional calls are explored in the next section.

Component	Port
edge-management-server	8080
edge-router	8081
edge-message-processor	8082
edge-qpid-server	8083
edge-postgres-server	8084

In addition to the apigee-service utility, Apigee also provides you with a service management API on Edge components.

The management API runs on custom ports on each of the components.

Valid API calls vary by component, but they all support a core set of APIs:

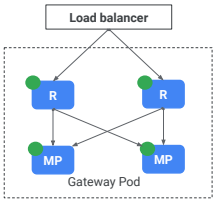
/v1/server/self returns statistics about that component.
/v1/server/self/up checks if the component is up and running.
/v1/service/self/uuid returns the UUID for that component.

You call the API using curl from the command-line or a HTTP client of your choice.

Maintaining Apigee Edge

Let’s now discuss some common maintenance operations that are performed on the platform.

Router and Message Processor accessibility



All R and MPs are reachable.

First, let's look at router and message processor accessibility.

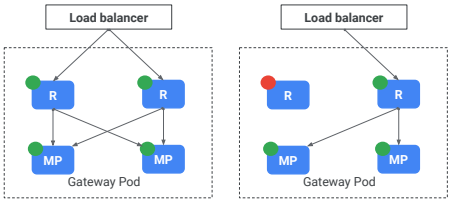
It is a good practice to disable access to a server during maintenance, such as for a server restart or upgrade.

When access is disabled, no traffic is directed to the server.

In the first example, we see a running system.

All components are reachable and traffic flows from the load balancer to all components.

Router and Message Processor accessibility



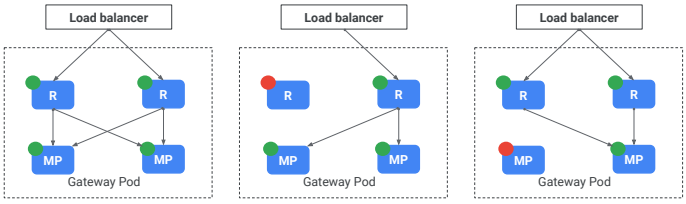
All R and MPs are reachable.

R access is disabled. Load balancer should stop sending traffic to the unreachable R. Load balancer configuration is done by the customer.

In the next example, access to one of the routers has been disabled.

This makes the load balancer send traffic only to the remaining router in the pool.

Router and Message Processor accessibility



All R and MPs are reachable.

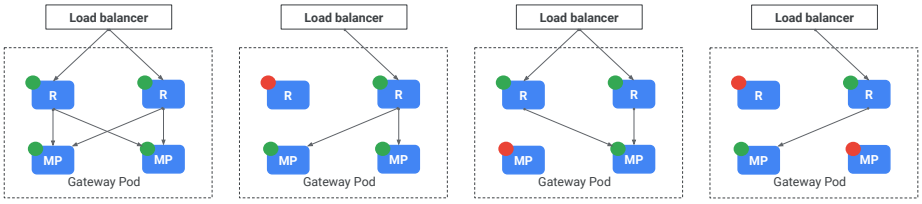
R access is disabled. Load balancer should stop sending traffic to the unreachable R. Load balancer configuration is done by the customer.

MP access is disabled. Both Rs will automatically stop sending traffic to the unreachable MP.

In the third example, access to one of the message processors has been disabled.

As a result, the routers automatically send traffic only to the accessible message processor.

Router and Message Processor accessibility



All R and MPs are reachable.

R access is disabled. Load balancer should stop sending traffic to the unreachable R. Load balancer configuration is done by the customer.

MP access is disabled. Both Rs will automatically stop sending traffic to the unreachable MP.

R and MP access is disabled. Only a single path exists, but the traffic still flows.

In this final example, access to one router and one message processor has been disabled.

Only a single path exists, but traffic still flows.

By following this approach, you allow maintenance activities on the platform while it continues to operate with minimal impact to users.

Router accessibility

Load balancers monitor port 15999 on the Routers to ensure that the route is available.

- To block the port, use the following iptables command on the Router node:
  `sudo iptables -A INPUT -i eth0 -p tcp --dport 15999 -j REJECT`
- To make the Router available, flush iptables using the command:
  `sudo iptables -F`
- If you use iptables to manage other ports on the node, use the -D option:
  `sudo iptables -D INPUT -i eth0 -p tcp --dport 15999 -j REJECT`

You use different approaches to disable routers and message processors.

In a production environment, you typically have a load balancer in front of the Edge Routers. You configure your load balancer to use port 15999 as the health check port for the routers.

If the port becomes unavailable, the router should be removed from the load balancing pool until it becomes available again.

To make the router unreachable, use the iptables command to block access to the port as shown.

Once maintenance is complete, reverse the change in order to allow the load balancer to reinstate the router back into the pool.

If you use iptables to manage other ports on the node, use the -D option when you flush iptables or use iptables to block the port.

Message Processor accessibility

- To disable access to a Message Processor, run the command below to stop it:
  `/opt/apigee/apigee-service/bin/apigee-service edge-message-processor stop`
- The Message Processor first processes any pending messages before it shuts down. Any new requests are routed to other available Message Processors.
- To restart the Message Processor, use the following commands:
  ```
  /opt/apigee/apigee-service/bin/apigee-service edge-message-processor start
  /opt/apigee/apigee-service/bin/apigee-service edge-message-processor wait_for_ready
  ```
- The wait_for_ready command returns the following message when the Message Processor is ready to process messages:
  `Checking if message-processor is up: message-processor is up.`

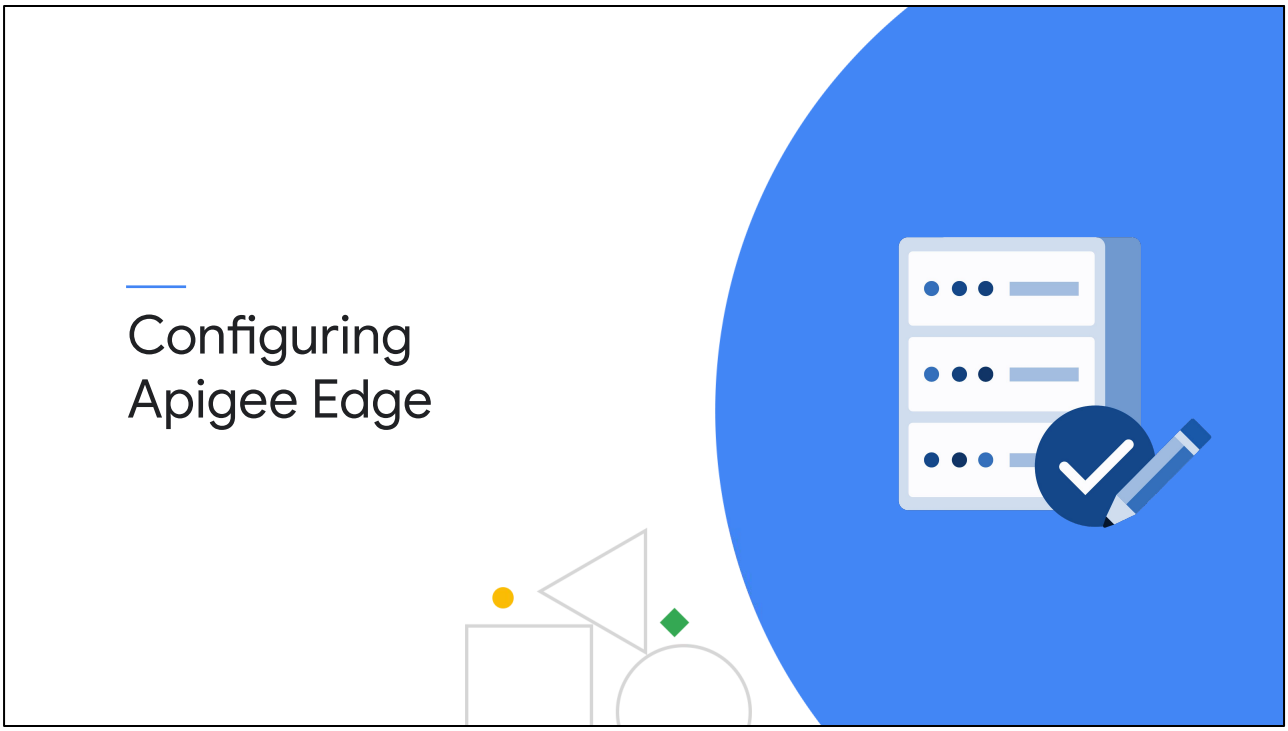
To disable access to a message processor, simply stop the service using the apigee-service utility.

The message processor will first process any pending messages before it shuts down. After this time, routers will send traffic to any remaining message processors.

To restart the message processor, use the apigee-service utility. You can also run the apigee-service utility with the wait_for_ready command.

When the message processor has finished starting up and is ready to process commands, you'll see a message stating that the service is up.

Configuring Apigee Edge



Let’s discuss how you can configure Apigee Edge for private cloud.

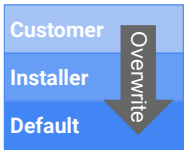
Code with config

- To configure components, Apigee Edge uses a combination of properties files and apigee-service actions.
- The technique of editing properties files is referred to as "code with config":
  - Consists of a key-value lookup tool
  - Keys are known as tokens
  - Token values are set in properties files
- Apigee Edge ships with default values that are overridden by the installer and by customers.
- Tokens are read from each location and merged into the final runtime configuration for each component.
  - Final configuration is written to `/opt/apigee/<component>/conf`

Customer
Customer overrides are written to `/opt/apigee/customer`

Installer
Installer overrides are written to `/opt/apigee/token`

Default
Apigee Edge ships with default values of component configuration.



To configure Apigee Edge, we use a process called code with config.

After installation, Edge uses a combination of properties files and apigee-service actions.

For example, to configure SSL on the management API, you edit the properties files to set the appropriate properties and then use the apigee service to restart the component and apply the changes.

This process of editing the properties files is referred to as "code with config."

It consists of a key-value lookup based on settings and the properties files. The keys are known as tokens. To reconfigure Edge, you set tokens in custom properties files.

This method allows Edge to ship with a set of default property values that can be overridden by both the installer and the customer.

Customer overrides are saved in the /opt/apigee/customer directory, while install time overrides are written to the /opt/apigee/token directory.

Code with config tokens are read from each location and merged into a final runtime configuration for each component.

Note that any customer property settings take priority over the installer settings. If no

overrides are found, the default settings are used to configure the component.

The final runtime configuration is available in the config directory of the component.

Properties files

- The /opt/apigee/customer/application directory contains customer-specific overrides.

  `/opt/apigee/customer/application/<component>.properties`

- Each component has a dedicated properties file:
  - postgresql.properties
  - management-server.properties
  - router.properties
- To set a component property, set the token to a value, and then restart the component.
- During an upgrade, the /opt/apigee/customer directory contents are never modified.

message-processor.properties

```
bin_setenv_min_mem=2048m
bin_setenv_max_mem=4096m
bin_setenv_meta_space_size=4096m
```

The /opt/apigee/customer/application directory is empty by default. The property file with settings that override the installer settings is created in this directory.

Each Edge component uses its own properties file that is named after the application without its prefix, for example: message-processor.properties or router.properties.

When you need to override a property, create the properties file and change its user and group ownership to the account that was used to install Apigee Edge.

To set a component property, edit the properties file and provide a value for the property's token. Once complete, save the file and use apigee-service to restart the component.

If you need to set a token value for a component for which a properties file does not currently exist, you must create the properties file.

Note that the files in the /opt/apigee/customer directory are not modified during the Edge upgrade process, so any customer property overrides are persisted.

Tokens

- All default token values are in the `/opt/apigee/<component>/token/default.properties` file.
- Before you override a component's token value, you should first determine its current value using:
  `/opt/apigee/apigee-service/bin/apigee-service <component> configure -search <token>`
- The command searches the hierarchy of properties files to determine the current value of the token.

  For example, to check the current value of the `conf_router_HTTP.request.line.limit` token for the router, run:

  ```
  /opt/apigee/apigee-service/bin/apigee-service edge-router configure -search
  conf_router_HTTP.request.line.limit
  ```

  The output of the command is:

  ```
  Found key conf_router_HTTP.request.line.limit, with value, 4k, in
  /opt/apigee/edge-router/token/default.properties
  ```

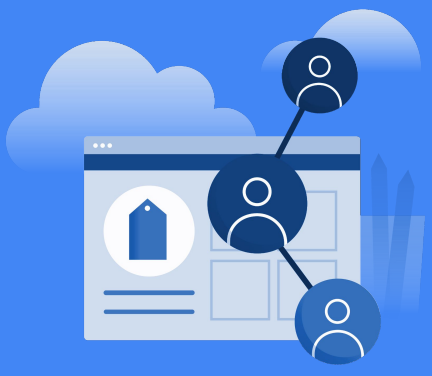
The default.properties file in the /opt/apigee/<component>/token directory includes the defaults that ship for all tokens. Any token here can be overridden.

Before overriding a token, you should determine the currently configured value by using the apigee-service utility and provide the component name, the configure command and the search option with the token that you want to inspect.

This command searches the properties file hierarchy to determine the current value of the token.

The example retrieves the current value of the HTTP request line limit token for the router component.

User management



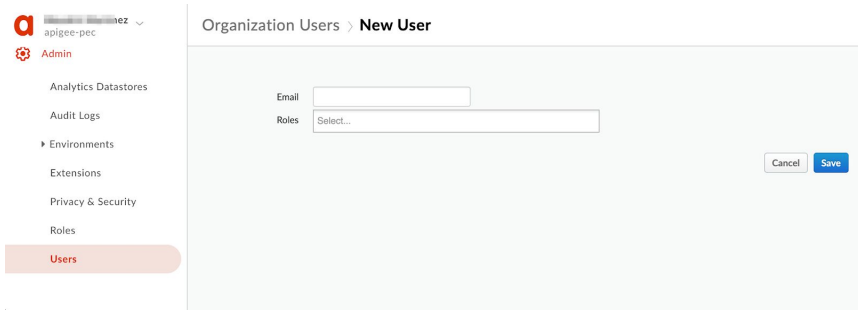
Now let’s talk about User Management.

This module discusses the various ways of managing your users in Apigee Edge for private cloud.

User management: Adding a user (Edge UI)

`http://<management-ui>:9000/login`

- Organization administrators, global sysadmin users, and users with appropriate permissions can add and remove users from an organization.
- Newly created users will receive an e-mail notification asking them to set up their initial password.



You can create a user by using the Apigee Edge UI, API, or Edge commands. The most convenient way to create a user is by using the Edge UI.

After you log in to the UI, under the Admin menu, click Users.

To add a new user, click add user and fill in all the details, selecting any roles that that user should have.

Newly created users receive an email with a link to reset their initial password.

Organization admins, global sysadmins, and users with appropriate privileges are the only roles that can add and remove users from an organization.

User management: Adding a user (management API)

Use the following command to create a user with the Edge API:

```
curl -iu <admin-email>:<password> -H content-type: application/json
-X POST http://<ms_IP>:8080/v1/users -d '{"emailId": "<email>",
"firstName": "<name>", "lastName": "<name>", "password": "<password>"}'
```

You can then use this API to view information about the user:

```
curl -u <admin-email>:<password> http://<ms_IP>:8080/v1/users/<email>
```

You can also add and remove users using the management API.

You add a user by posting a JSON payload with the new user details to the management API /v1/users.

For the API to work correctly, you need to supply admin credentials.

It is also possible to call this API using an XML payload.

Users are identified by their email address. To retrieve user information, invoke the same API with a GET request and append the user's email address to the request URL.

User management: Adding a user (apigee-service)

Users can also be added by using apigee-service:

```
/opt/apigee/apigee-service/bin/apigee-service apigee-provision create-user
-f <config>
```

Sample configuration file:

```
# If APIGEE_ADMINPW is omitted, you will be prompted.

APIGEE_ADMINPW="Password"
USER_NAME="foo@bar.com"
FIRST_NAME="New"
LAST_NAME="User"
USER_PWD="UserPassword"
```

Some administrators may choose to use the apigee-service utility to add users.

This can be done by creating a simple configuration file that contains the user information.

The user is then created using the apigee-service utility with the apigee provision create-user command and supplying the path to the config file.

Note that the config file should be readable by the apigee user.

User management: User roles

Assign the user to a role in an organization:

```
curl -iu <admin-email>:<password> -H
“content-type:application/x-www-form-urlencoded” -X POST
http://<ms_IP>:8080/v1/o/<org>/userroles/<role>/users?id=<email>
```

View the user's roles:

```
curl -iu <admin-email>:<password>
http://<ms_IP>:8080/v1/users/<email>/userroles
```

Remove a role from a user:

```
curl -iu <admin-email>:<password> -X DELETE
http://<ms_IP>:8080/v1/o/<org>/userroles/<role>/users/<email>
```

Before a user can perform any actions, they need to be assigned a role in the organization.

Example roles include org admin, business user, ops admin, or a custom role that is defined for your organization.

To add a user to a role in the organization using the /userroles management API, you invoke the /users endpoint, and provide the user's email address as the value of the ID parameter,

The role that a user belongs to can be retrieved using a GET operation to the /userroles endpoint of the /users management API, and providing the user's email address.

To remove a user from a role, you perform a DELETE operation to the /userroles management API’s /users endpoint and provide the user’s email address.

Note that if you're removing a user from an organization, they must first be removed from all roles.

User management: Sysadmin users

- The global sysadmin role exists outside of organization boundaries.
- It has general administrative access over the entire system, including all organizations.
- There can be more than one sysadmin user.
- Best practice is for users with the sysadmin role to not belong to any organization.
- You cannot use the Edge UI to assign a user to the sysadmin role. This can only be done using the management API.

Apigee Edge has a special type of user, the sysadmin user. During the installation process, a single sysadmin user is created.

The global sysadmin role exists outside of the organization boundary, and has general administrative access over the entire system including all organizations and components.

Users assigned to the sysadmin role can add and remove capacity by adding and removing hardware, change system configuration, create organizations and environments, and perform many other system-wide administrative functions.

There can be more than one sysadmin user. Any user who is a member of the sysadmin role has full permissions to all resources.

Users with the sysadmin role are not required to be part of an organization, and it's a best practice for such a user to not belong to one. However, only users with membership in an organization can log in to the Edge UI.

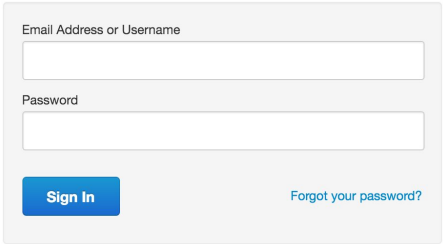
You can create the system administrator user using either the Edge UI or management API, but you must use the API to assign the user to the sysadmin role. Assigning a user to the sysadmin role cannot be done via the UI.

To add a user to the sysadmin role, invoke the /userroles/sysadminusers/ management API and provide the user's email address as the value of the ID

parameter.

User management: Password reset

- After the installation, you can reset the passwords for Openldap, the Edge primary sysadmin user, the Edge organization user, and Cassandra.
  - End users can perform self-service password resets using the “Forgot your password?” link.



https://docs.apigee.com/private-cloud/v4.50.00/resetting-passwords

User passwords can be reset in multiple ways.

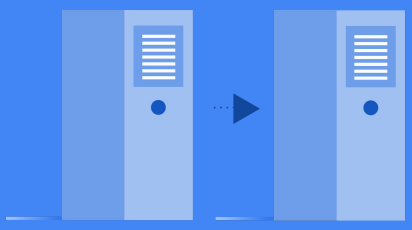
Using the apigee-service utility, you use the apigee-setup functionality to reset passwords, providing the username, the new password, and sysadmin credentials.

This can also be done by providing a config file to the utility.

Refer to the documentation at the link provided for detailed instructions on resetting passwords for different Edge components.

Users can also self-serve by using the "Forgot Your Password?" link on the Apigee UI login page.

Backup and Restore



Let’s now discuss the backup and restore process for the Apigee private cloud platform.

Backup and Restore: Considerations

Backup

- Cassandra: Back up Cassandra nodes one at a time.
- ZooKeeper: Same considerations as for Cassandra. The backup process temporarily shuts down ZooKeeper.
- PostgreSQL: If your installation contains more than one PostgreSQL node, back up the nodes one at a time or back up only the active instance.

Restore

When you restore one of the ZooKeeper, Cassandra, or Openldap nodes, it is recommended to restore all the nodes in order to achieve consistency.

It is strongly recommended to perform backups across all components in all data centers while they have the same configuration state.

Apigee has made the backup and restore process easy to execute, but there are certain aspects of the system you should consider.

Cassandra nodes should be backed up one at a time. The backup process temporarily shuts down Cassandra, so you do not want to run it at the same time for all Cassandra nodes.
Performing a rolling backup allows for adequate Cassandra availability in an installation of 3 or more Cassandra nodes.

Zookeeper - has the same considerations as those for Cassandra. The backup process temporarily shuts down ZooKeeper.

PostgreSQL - If your installation contains more than one PostgreSQL node, back up the Postgres nodes one at a time. This will eliminate any impact to the UI. In a PostgreSQL active/standby configuration, you may choose to only backup the active instance. After it is restored, you can add another standby node and populate it via replication.

When you restore one of the ZooKeeper, Cassandra, or Openldap nodes, it is recommended to restore all nodes in the cluster in order to achieve consistency (specially when organizations or environments have been created since the last

backup).

Backup

Use the following command to perform a backup:

```
/opt/apigee/apigee-service/bin/apigee-service <component> backup
```

For example:

```
/opt/apigee/apigee-service/bin/apigee-service apigee-cassandra backup
```

As mentioned earlier, performing a backup is relatively simple.

Use the apigee service utility with the backup command option and provide the component name.

For example, apigee-service apigee-cassandra backup will back up the Apigee Cassandra component.

Backup command actions

- Stops the component (except for PostgreSQL, which must be running to back up).
- Creates a tar file of the following directories and files:
  - Directories
    - `/opt/apigee/data/<component>`
    - `/opt/apigee/etc/<component>.d`
  - Files (if they exist)
    - `/opt/apigee/token/application/<component>.properties`
    - `/opt/apigee/customer/application/<component>.properties`
    - `/opt/apigee/customer/defaults.sh`
    - `/opt/apigee/customer/conf/license.txt`
- Writes the tar file to `/opt/apigee/backup/<component>`. The file name is in the form: `backup-(year).(month).(day),(hour).(min).(seconds).tar.gz`.
- Restarts the component.

The backup command performs the following actions:

It stops the component, except for PostgreSQL, and creates a tar file of the component's **data** and **etc** directories.

If the component.properties file exists in the /opt/apigee/token/application or /opt/apigee/customer/application directories, they are also backed up along with the defaults.sh and license.txt files.

The backup process creates a directory using the components name in the /opt/apigee/backup directory if it does not already exist.

The backup tar file is then written to that directory using the year.month.day and time in the filename.

The backup command then restarts the component if required.

To further simplify the process, all components on a node can be backed up using the apigee-all backup command.

Restore

To perform a restore:

```
/opt/apigee/apigee-service/bin/apigee-service <component> restore <backup-file>
```

For example:

```
/opt/apigee/apigee-service/bin/apigee-service apigee-cassandra restore
backup-2020.11.19,14.40.41.tar.gz
```

The backup file is optional. If omitted, the restore command uses the latest file in `/opt/apigee/backup/<component>`

A restore of the platform can only be done one component at a time.

Unlike backups, you cannot use apigee-all to restore all components on a node.

To restore a component, you use the apigee-service utility with the restore command option and provide the component name and an optional backup file from a previous backup operation.

If no backup file is supplied, the utility uses the file created from the most recent backup.

In the example, the apigee cassandra component is being restored from a previous backup in November 2020.

Restore command actions

The restore command:

- Uses the specified backup file or the latest backup file, if a filename was not provided
- Checks to see if these directories are empty:
  - `/opt/apigee/data/<component>`
  - `/opt/apigee/etc/<component>.d`
- Stops and prompts you to remove the above directories if not empty
- Stops the component
- Restores the component from the backup

Action required:
You must explicitly restart the component after a restore.

The restore command performs the following actions:

It uses the specified backup file or the latest backup if no file was specified.

It checks to see if the /opt/apigee/data and /opt/apigee/etc component directories are empty.

If these directories are not empty, the restore operation fails and you are prompted to remove them.

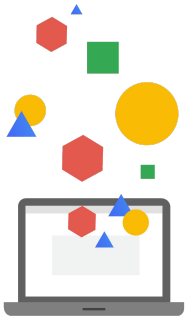
If the directories are empty, it stops the component, and the backup is restored.

Restore does not restart components automatically, you must do that manually.

Data migration tools

All data in Apigee Edge is accessible via APIs, so you can develop your own tools or use third-party tools.

Apigee Organization Data Migration Tool:
https://github.com/apigeecs/apigee-migrate-tool



Some customers may need to migrate data between organizations that are in the same or different planet in Apigee Edge.

You can use an open source organization data migration tool to assist with this task.

You are not limited to using this tool. Since all Apigee Edge data is accessible via APIs, you can develop your own tools, enhance this tool for your own purposes, or use third-party tools.

Agenda

Platform operation

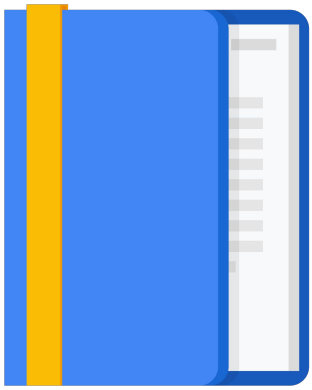
Infrastructure operation

REST fundamentals

Management API

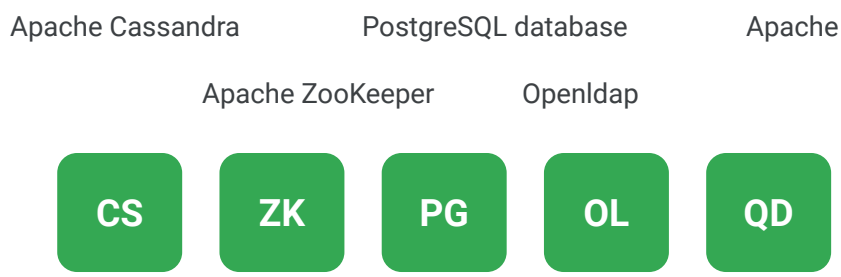
API proxy deployment

Edge analytics



In this lesson, we explore the management of the open source components used in Apigee for private cloud.

Open source components



Apigee Edge uses multiple leading open source technologies.

Apigee does not modify the open source software. If fixes or improvements are needed, Apigee works with the open source community and contributes to it.

You can refer to any available documentation for these products and use applicable third-party tools that are compatible with Apigee open source product distributions.

In this section, we explore the management of the open source components used in private cloud, namely, Apache Cassandra, Apache ZooKeeper, and Apache Qpid.

Cassandra: Nodetool informational commands

Check ring status:
```
nodetool -h0 ring
```

Check ring status:
```
nodetool -h0 status
```

Check compaction status:
```
nodetool -h0 compactionstats
```

Check node info:
```
nodetool -h0 info
```

Check Gossip info:
```
nodetool -h0 gossipinfo
```

Status of the thrift server:
```
nodetool -h0 statusthrift
```

Check reads/writes/drops:
```
nodetool -h0 tpstats
```

Displays statistics for every keyspace and column family:
```
nodetool -h0 cfstats
```

Let's discuss the administration utility nodetool and see how it is used to manage the open source component Apache Cassandra.

Nodetool is an administration utility provided by Cassandra, and it is located in the /opt/apigee/apigee-cassandra/bin directory.

To use the tool, you specify the host on which to perform an action and the action or command to run. The examples shown use -h0, which indicates the local host.

nodetool ring displays the status of the Cassandra ring, as seen from the perspective of the node being queried. Note that if your cluster is incorrectly configured, you may see different outputs if you run the command on different nodes.

nodetool status returns information about the cluster, such as state, load, token range, data ownership, rack information, and IDs of member nodes.

nodetool compactionstats returns the name of any SS tables that are currently being compacted, the progress of each compaction event, and the time remaining.
This is useful if you need to confirm whether any observed high disk I/O is being caused by compaction events.

nodetool info provides information for a single node, such as token and storage information, and uptime and heap memory usage.

nodetool gossipinfo provides gossip information for the cluster. gossip is the internode communication protocol used by Cassandra.

nodetool statusthrift provides thrift information. Thrift is the low-level API that clients use to communicate with Cassandra.

nodetool tpstats provides statistics on the number of active, pending, and completed tasks for each of the Cassandra thread pools.
An output from tpstats that contains a large number of active threads in the hinted handoff pool could indicate a connectivity issue in the ring.

nodetool cfstats provides statistics on keyspaces and column families. Interesting statistics include read and write latency, memtable size, size on disk, to name a few .
Use this command with the -H option to display more human-readable information.

Cassandra: Nodetool operational commands

Stop compactions:

```
nodetool -h localhost stop COMPACTION
```

Disable Thrift:

```
nodetool -h localhost disablethrift
```

Disable Gossip:

```
nodetool -h localhost disablegossip
```

Flush all memtables and stop listening for client requests until the node is restarted:

```
nodetool -h localhost drain
```

Flush all memtables from the node to SSTables on disk:

```
nodetool -h localhost flush
```

Use with caution!

Note that the commands discussed here should be used with caution.

nodetool stop compaction stops a compaction event. You typically run this command if it's been determined that compaction events are having a negative impact on node performance.

When compaction stops, Cassandra will continue working through any pending operations in its queue and eventually restart the stopped compaction.

nodetool disable thrift prevents a node from acting as a coordinator.

Because the node can still serve read requests, you would normally also run the nodetool disable gossip command. At this point, the node is effectively marked as unavailable in the cluster.

These commands may be used if you're performing maintenance or troubleshooting the node.

The drain and flush commands flush memtables to SSTables on disk.

nodetool flush simply flushes the memtables, while nodetool drain flushes the memtables and also stops listening for further client requests until the node is restarted.

ZooKeeper: Four-letter commands

- Apache ZooKeeper has several "four-letter commands."
- Commands are sent as text to a network port.

Example:

```
echo ruok | nc localhost 2181

imok
```

Example:

```
echo stat | nc localhost 2181

Latency min/avg/max: 0/0/26
Connections: 8
Outstanding: 0
Zxid: 0x1000007e0
Mode: follower
Node count: 470
```

ZooKeeper status can also be obtained with:

```
/opt/apigee/apigee-zookeeper/bin/zkServer.sh status
```

Apache ZooKeeper is another open source component used in Apigee for private cloud.

ZooKeeper has several four-letter commands that can be used to query its reachability and its current status with respect to its follower, leader, and observer nodes.

A full list of the four-letter commands is provided at the link to the ZooKeeper's Administrator guide.

The commands can be invoked using the netcat command, Telnet, or any other utility that can send commands to a specific TCP port.

A commonly used 4-letter command is ruok (aRe yoU OK). Echoing ruok and netcatting it to port 2181 should return the response "imok" if the service is up and running.

For more information about the ZooKeeper service, you can use another command called stat, that displays statistics on the ZooKeeper service some of which are shown here.

The sample output shows that the maximum latency is 26 milliseconds, there are 470 nodes in the ZooKeeper hierarchy, and the node being queried is currently a follower.

Qpid

Qpid statistics can be retrieved using the command:

```
qpid-stat -q
```

OUTPUT:

```
Queues
  queue                                     dur  autoDel  excl  msg   msgIn  msgOut  bytes  bytesIn  bytesOut  cons  bind
  ============================================================================================================================
  581990aa-5f25-4044-8467-d08b0b7dd447:0.0       Y        Y        0     0      0       0      0        0         1     2
  ax-q-axgroup-001-consumer-group-001       Y                      0     90     90      0      9.55m    9.55m     12    2
  ax-q-axgroup-001-consumer-group-001-dl    Y                      0     0      0       0      0        0         0     2
```

To check queue depth, run the command:

```
python check_queue_depth.py
```

OUTPUT:

```
QueueName: ax-q-axgroup-001-consumer-group-001
queue depth: 0
```

Our final open source component is Apache Qpid.

Several utilities that interact with Apache Qpid are located in the /opt/apigee/apigee-qpidd/bin directory.

The two most useful commands are described here.

The first is qpid stat dash q. This command displays all the configured message queues used by Qpid with statistics for each queue.

In a healthy analytics system, the value of msgIn should equal that of msgOut, and the same should be true for bytesIn and bytesOut.

Another method of retrieving similar information is to use the Python script, check_queue_depth.py, which lists the name of each analytics queue and its queue depth.

You would typically use this script to confirm that a queue has been drained before performing maintenance on the node.

Components maintenance documentation

Recurring Edge Services maintenance tasks:
https://docs.apigee.com/private-cloud/latest/recurring-edge-services-maintenance-tasks

The Apigee Recurring Edge Services maintenance tasks documentation provides additional information regarding the management of all Edge components.

The guide includes typical commands, recurrent activities, and other actions and is available at the link provided.

Agenda

Platform operation

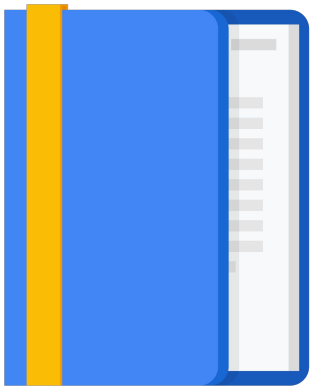
Infrastructure operation

REST fundamentals

Management API

API proxy deployment

Edge analytics



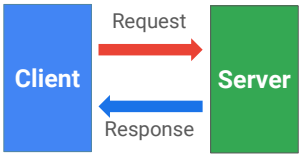
Let’s discuss some high level concepts of REST and the HTTP protocol that is used by REST APIs.

HTTP: Overview

- The Hypertext Transfer Protocol (HTTP) is an application protocol.
- HTTP is the protocol to exchange or transfer hypertext.
- Communication between a client and a server occurs via a request/response pair.
- HTTP specification: https://www.w3.org/Protocols/rfc2616/rfc2616.txt.

Uniform Resource Locator (URL)

```
http://<host>:<port>/<path to resource>?<params-name>=<param-value>
```



The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

HTTP uses a request - response mechanism to communicate between the client and the server.

The full specification of the protocol is described in RFC2616, available on the w3 website.

When describing the full address of a resource via the HTTP protocol, we often specify a URL.

A URL consists of:
The protocol prefix (http or https) that specifies whether the connection is in plaintext or encrypted.
The host - a Fully Qualified Domain Name (FQDN) or IP address of the server providing the resource.
The port - the TCP port on which the server expects connections.
The path to the resource - a string defining the location of the resource on the server, similar to a directory structure.

Optional query parameters, which are key-value pairs that contain data that is sent along with the request.

Note that query parameters are part of the request URL and are logged in the access logs of the server. They should not be used to contain PII or other sensitive information (like credentials).

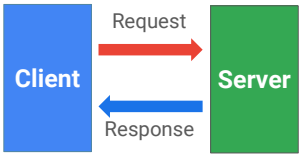
HTTP: Overview

HTTP methods:

- GET: Fetch an existing resource.
- POST: Create a new resource.
- PUT/PATCH: Update an existing resource.
- DELETE: Delete an existing resource.
- HEAD: This is similar to GET, but without the message body.
- TRACE:Used to retrieve the hops that a request takes to and from the server.
- OPTIONS: Used to retrieve the server capabilities.

Status Codes:

HTTP/1.1: Status Code Definitions



The HTTP protocol supports several methods:

GET - to fetch an existing resource.
POST - to create a new resource.
PUT/PATCH - to update an existing resource.
DELETE - to delete an existing resource.
HEAD - This is similar to GET, but without a message body.
TRACE - is used to retrieve the hops that a request takes to and from the server.
OPTIONS - is used to retrieve server capabilities.

Status codes make up part of the response from the server and are used to indicate the status of the request.

They consist of 3-digit numbers and are grouped into ranges to indicate information, success, redirection, client errors, and server errors status.

The full list and description of the http status codes is available at the link provided.

HTTP: Request and response message formats

Message format:

- A Start-line
- Zero or more header fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF)
  - indicates the end of the message headers
- An optional message body

Message header:

- Message header = field-name ":" [field-value]
- Example:
  - Content-Type: application/json
  - User-Agent: Mozilla/5.0

Message body

- If message body is present, then usually Content-Type and Content-Length headers specify the format and length of the request payload.

A HTTP message consists of:

A start line that indicates the beginning of the message,

Zero or more header fields,

An empty line that indicates the end of the header fields and

An optional message body, also called a “payload,” which is sent as part of the request.

Message headers are in the form of key-value pairs that include a field name and field value that are separated with a colon character.

The message body, if present, is usually accompanied by a content-type header that indicates the format of the payload.

REST: Overview

- Representational state transfer (REST) is the software architectural style of the World Wide Web.
- REST was defined by Roy Thomas Fielding in his 2000 PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures" at https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Resource	POST	GET	PUT/PATCH	DELETE
/dogs	Create new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/bo	Error	Retrieve Bo	If exist, update Bo. If not, error	Delete Bo

REST is an acronym for representational state transfer.

REST uses URIs to refer to and access resources.

REST is built on top of the stateless HTTP protocol and uses HTTP methods to define operations on a resource.

REST: APIs

Resource structure

```
GET /owners/5678/dogs
```

Queries

```
GET /dogs?color=red&state=running&location=park
```

Partial response

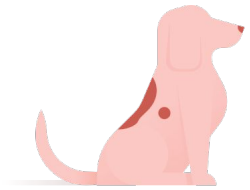
```
GET /owners/5678/dogs?fields=id,name,picture
```

Pagination

```
GET /dogs?limit=25&offset=50
```

API versioning

```
/dogs/v1
```



REST is an architectural style and provides a standard to structure an API.

Client applications interact with the API over HTTP by making requests to specific URLs, and often getting relevant data back in the response.

In order to use a http method on a resource, you should be aware of its path structure. The examples show two resources: owners and dogs.

To fetch a list of dogs that belong to a specific owner, you use the GET http method with the /owners/owner-id/dogs url path.

You can also filter the response from the API using query parameters. The example fetches a list of all dog resources filtered by the color, state, and location of the dogs.

By default, an API may return the full set of resource attributes in its response. You can request a partial response to limit the size of the payload by using a query parameter in the request to specify the list of attributes required.

The example uses the fields query parameter to limit the set of attributes to only include the ID, name, and picture of each dog resource in the response.

You can also implement pagination of the API response by using limit and offset query parameters.

It is a best practice to version your APIs . Each API has a specific “contract” that includes the resource url structure and data, so it is a good idea to release a new version of the API if the contract changes. This ensures backward compatibility of the API with client applications and a path to deprecation of the older version of the API.

JSON: Overview

- JSON is a language-independent data format.
- JSON home - http://www.json.org

Syntax specification

```
object
   {}
   { members }
members
   pair
   pair , members
pair
   string : value
array
   []
   [ elements ]
elements
   value
   value , elements
value
   string
   number
   object
   array
   true
   false
   null
```

Example

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd
Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON, or JavaScript Object Notation, is a lightweight data-interchange format that is based on a subset of the JavaScript Programming Language.

It has become a standard for exchanging data using REST APIs.

JSON uses a string format to specify basic data types like strings, numbers, arrays, booleans, and other object literals.

You can also nest JSON objects to construct a data hierarchy.

curl

- curl is an open source command-line tool and library for transferring data with URL syntax.
- It supports HTTP and HTTP/2 among many other network protocols.
- Distribution: https://github.com/curl/curl

```
curl https://retailstore.com/v1/categories

[{"color":"#506d7d","id":0,"name":"Appliances"},{
"color":"#94cbb9","id":1,"name":"Automotive"},{"c
olor":"#ffc20e","id":2,"name":"Baby"},{"color":"#
f7a969","id":3,"name":"Books"},{"color":"#6c507d"
,"id":4,"name":"Cameras"},{"color":"#f07d65","id"
:5,"name":"Clothing"},{"color":"#e36b22","id":6,"
name":"Electronics"},{"color":"#b34b37","id":7,"n
ame":"Fitness"},{"color":"#97779e","id":8,"name":
"Gifts"},{"color":"#71a973","id":9,"name":"Health
"},{"color":"#b75f5f","id":10,"name":"Home"}]
```

```
curl -v http://google.com
* TCP_NODELAY set
* Connected to google.com (172.217.169.110) port 80 (#0)
> GET / HTTP/1.1
> Host: google.com
> User-Agent: curl/7.64.1
> Accept: */*
< HTTP/1.1 301 Moved Permanently
< Location: http://www.google.com/
...
< Content-Length: 219
< X-XSS-Protection: 0
< X-Frame-Options: SAMEORIGIN
<HTML><HEAD><meta http-equiv="content-type"
content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

Curl is an open source command line tool and library that is used for transferring data using URL syntax. It is included by default in most modern linux distributions.

It supports various protocols such as FTP, HTTP, IMAP, and LDAP.

Curl also supports SSL certificates, HTTP form based upload, proxies, HTTP/2, cookies, various forms of authentication, file transfer, and much more.

It is a very useful tool in the operations developer toolbox.

curl: Sample commands

Perform a GET on specified URL:

```
curl -v http://www.apigee.com
```

Specify the username and password for server authentication:

```
curl -u <user>:<password> http://<host>:<port>/<resource>
```

Perform a POST on the specified resource, with header and data:

```
curl -u <user>:<password> -H "<header>" -X POST http://<host>:<port>/<resource> -d
'<data>'
```

Perform a DELETE on the specified resource <id> with header:

```
curl -u <user> -H "<header>" -X DELETE http://<host>:<port>/<resource>/<id>
```

Here are a few examples of curl usage.

By default curl uses the GET method, deriving the protocol from the URL. The output returned from the command is the response payload received from the server.

Adding the -v or verbose option instructs curl to include in the output the request and response headers in addition to other information.

If the URL requires authentication, you can use the -u option to provide credentials in username, colon, password format. If you omit the password, you'll be prompted for it on the terminal.

To use a different HTTP method, specify the -X option followed by the http verb.

Headers are added by using the -H option, followed by the header field, a colon, and the header value. It is a good practice to wrap header field:value pairs in quote characters.

Agenda

Platform operation

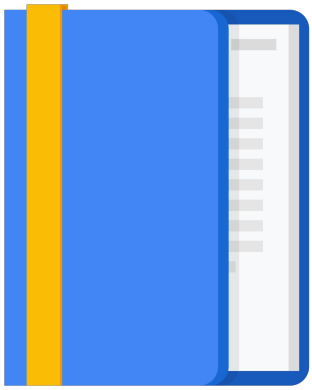
Infrastructure operation

REST fundamentals

Management API

API proxy deployment

Edge analytics

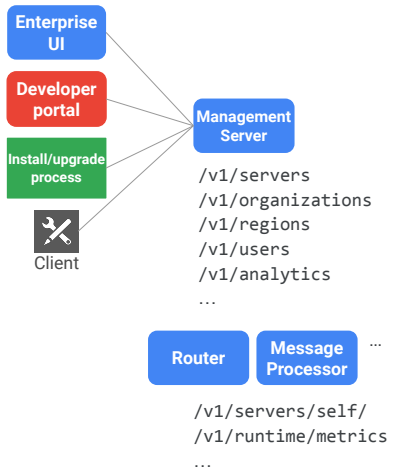


In this lesson, we'll cover an overview of the management API, supported resources, and tools that you can use to interact with the API.

Management API overview

- Apigee Edge provides management interfaces based on REST APIs.
- The Management Server exposes a centralized unified REST API called the Management API.
- Enterprise UI operations are all powered by the Management API.
- Authentication to the management API is via HTTP basic auth or OAuth2.

https://apidocs.apigee.com/apis



Apigee for private cloud provides management interfaces based on REST APIs.

Each server level API provides basic functions related to the server status and operation.

The example shows router and message processor server APIs, which report on their health and provide run-time metrics.

The Management Server exposes a centralized unified REST API called the Management API.

The Management API governs all functions on Apigee Edge, from operations to deploy API proxies to operations that add and remove capacity to and from a given pod, environment, or organization.

The API supports JSON and XML and is secured by default via HTTP Basic Authentication. It is possible to enable OAuth2 authentication by installing the apigee-sso component.

Enterprise UI operations are all powered by the management API, with data returned from the management server.

The Apigee private cloud Installation and upgrade process and management scripts make use of the management API.

More details on the API are available at the link provided.

Management API basics

General structure:

```
/v1/<resource>/<id>?<name>=<value>&<name>=<value>
```

Examples:

```
/v1/organizations

/v1/organizations/<org-name>

/v1/organizations/<org-name>/environments

/v1/organizations/<org-name>/environments/<env-name>

/v1/servers?pod=<pod-name>&region=<region-name>
```

The general structure of the management API includes the version number /v1 followed by a named resource, the resource name or ID and an optional set of query parameter name value pairs.

An example of a management API is /v1/organizations. This API returns a simple structure that contains a list of organizations.

If a specific organization name is provided in the API URL, the response payload contains information that describes that organization.

To retrieve a list of all the Apigee environments that are defined for an organization, use the environments endpoint.

To get specific details of an environment, append the name of the environment to the URL.

To get a list of servers defined in a given pod and region, use the /v1/servers API and provide the name of the pod and region as query parameters.

Management API: Supported resources

- Analytics
- API proxies
- API products
- Apps
- Apps: Company
- Apps: Developer
- App keys: Company
- App keys: Developer
- Audits
- Cached logs
- Caches
- Companies
- Company developers
- Data masks
- Debug sessions
- Deployments
- Developers
- Environments
- Extensions
- Key-value maps (KVMs)
- Keystores and truststores
- LDAP resources
- Mock target
- Monetization APIs
- OAuth 1.0a access tokens
- OAuth 1.0a request tokens
- OAuth 1.0a verifiers
- OAuth2 access tokens
- OAuth2 authorization code
- OAuth2 refresh tokens
- Organizations
- Policies
- References
- Reports
- Resource files
- Shared flows and flow hooks
- Stats
- Target servers
- User roles
- Users
- Virtual hosts

The list above represents a subset of resources.
Learn more about Management API by visiting
https://apidocs.apigee.com/operations

You can manage a wide set of Apigee resources using the management API.

These include infrastructure, proxy and configuration resources.

You can learn more about using the management API for each of the supported resources at the link provided.

Management API tools

Apigee Management API Postman collection
https://github.com/apigee/apigee-management-api-postman

Apigee Node tool
https://github.com/apigee/apigeetool-node

Apigee deploy, config Maven plugins
https://github.com/apigee/apigee-deploy-maven-plugin
https://github.com/apigee/apigee-config-maven-plugin

Apigee Python deploy tool
https://docs.apigee.com/api-platform/deploy/deploying-proxies-command-line

Apigee has developed and open sourced several tools for the Management API. These tools enable you to execute common actions such as deploy and undeploy APIs, using products that you may be more familiar with.

The Apigee Management API Postman collection includes a large number of commonly used API calls that can be used with the Postman REST client.

The Apigee tool is an npm module that includes functionality for the management of API proxies, KVM and cache management, and API products and developer management.

The apigee-deploy-maven-plugin is a build and deploy utility for building and deploying API proxy bundles onto the Apigee private cloud platform using maven.

The configuration maven plugin can be used to create and manage Apigee configuration objects such as Caches, KVMs, and Target Servers.

Both these tools can be used as part of your CI/CD process to aid in API proxy development, testing, and deployment.

If you use Python for development, the Apigee python deploy tool helps you manage your API proxy deployments.

Agenda

Platform operation

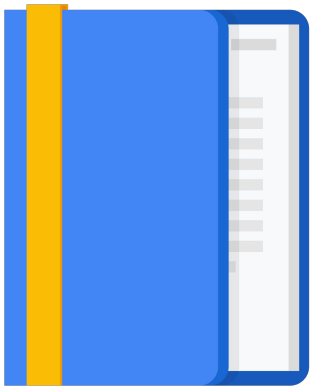
Infrastructure operation

REST fundamentals

Management API

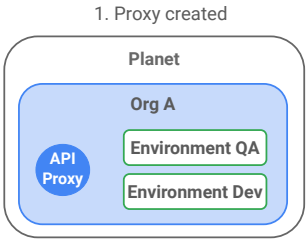
API proxy deployment

Edge analytics



In this lesson, we discuss API proxy deployment, importing and exporting API proxies, and SDLC integration.

Deploying an API proxy



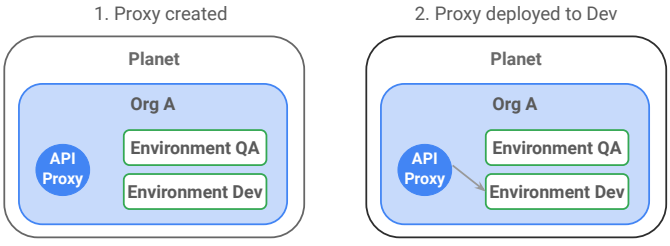
- An API proxy belongs to an organization.
- When an API proxy is created or imported, it is logically placed within the boundary of the specified organization.

An API proxy is a resource that belongs to an organization.

When you create or import a proxy, it is placed logically within the boundary of the organization you specify.

At this point, the proxy exists but cannot be used until it is deployed.

Deploying an API proxy



- An API proxy belongs to an organization.
- When an API proxy is created or imported, it is logically placed within the boundary of the specified organization.
- An API proxy deployment is a logical association between the API proxy revision and a specified environment.
- A revision represents a snapshot of the proxy code.

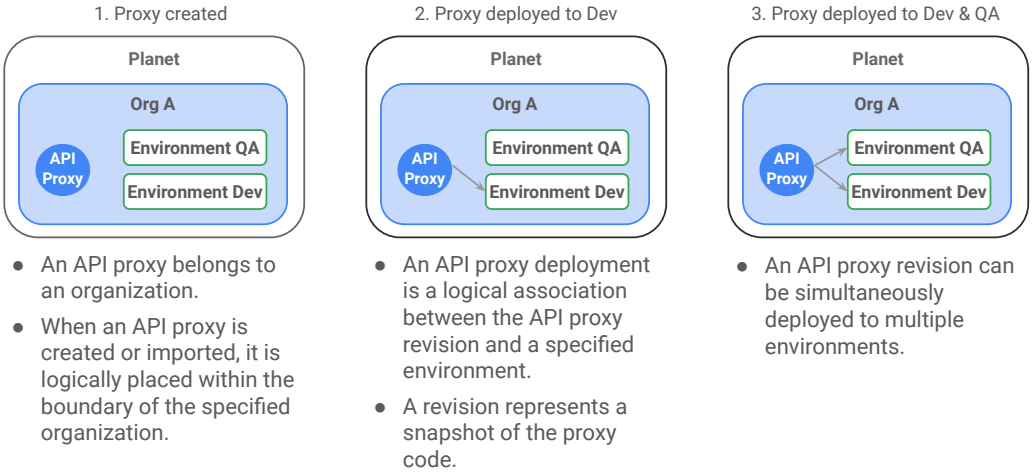
In the example, we have an environment called dev and an environment called QA.

When we deploy an API proxy to the dev environment, we create a logical association between the API proxy revision and the environment.

A revision is simply a snapshot of the proxy code.

At this point, the proxy is available to process API traffic to the dev environment.

Deploying an API proxy



- An API proxy belongs to an organization.
- When an API proxy is created or imported, it is logically placed within the boundary of the specified organization.
- An API proxy deployment is a logical association between the API proxy revision and a specified environment.
- A revision represents a snapshot of the proxy code.
- An API proxy revision can be simultaneously deployed to multiple environments.

As part of our API development lifecycle, we may also want to deploy the API proxy to the QA environment.

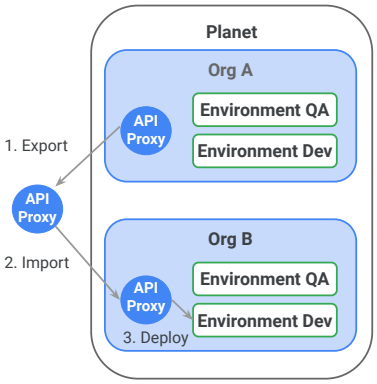
The same API proxy revision can be deployed across multiple environments in the same organization.

If you need to make changes to the API proxy, you can update it or re-import it to the organization.

Re-importing a proxy that already exists in the organization saves the proxy with a new revision number.

Deploying an API proxy to a new organization

- An Organization represents a tenant.
- In order to deploy an API proxy owned by org A to the Dev environment owned by Org B, the API Proxy must be:
  - Exported from org A
  - Imported to org B
  - Deployed to the Dev env in org B
- This applies to organizations created in the same or separate planet.



In Apigee Edge, an organization represents a tenant. Organizations are logical boundaries that implement the separation of resources.

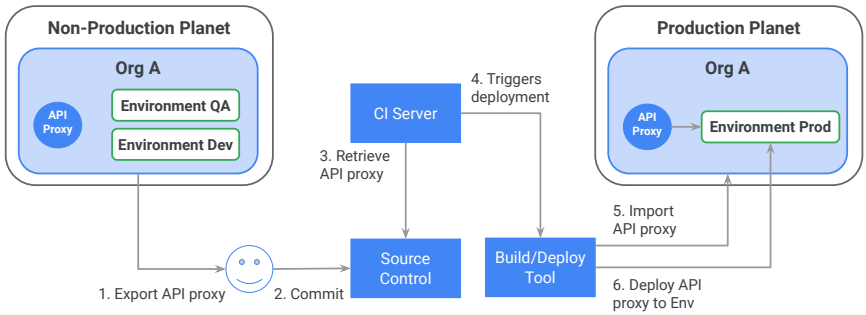
For this reason, to deploy a proxy owned by Org A to Org B, we need to first export that proxy from Org A, import the proxy to Org B, and then deploy the proxy to an environment in Org B.

The process applies to organizations regardless of whether they reside within the same or separate Apigee Edge planet or installation.

If the API proxy requires additional resources such as KVMs, virtual hosts, target servers, or other configuration, these objects must be created in the target organization and environment before the proxy deployment.

Integrating API proxy deployment with your SDLC

- Building, deploying, and configuring API proxies and resources is automatable.
- You can leverage your existing automation tools to manage API proxies and resources.



It is important to remember that a deployment may be composed of more than just API Proxies.

Target Servers, Cache resources, KVM definitions, API Products, and other config objects may also need to be promoted between organizations and environments.

Automating API Proxy deployment and other tasks related to Apigee management doesn't require new tools. You can leverage your existing tools such as Git, Jenkins, or Maven to integrate API Proxy management in your SDLC processes.

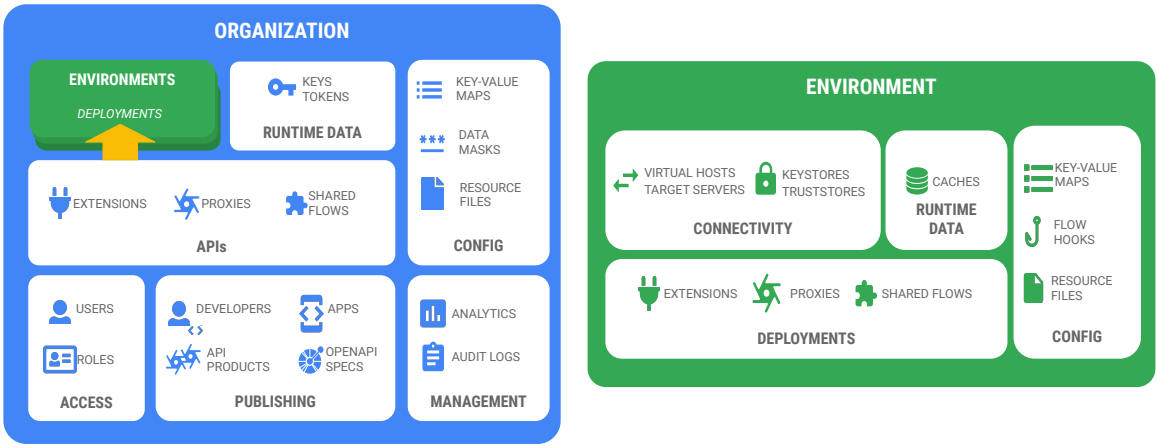
Here is an example that shows how CICD tools can be used with Apigee.

An API proxy and any dependent configuration is exported from the non-production planet or Apigee Edge installation.

It is then committed to source control, from where it is retrieved by the continuous integration server. This triggers a deployment, which is managed by the build or deploy tool.

After all the necessary integration tests are successfully run, the deploy tool first imports the API proxies into the production organization and then deploys the proxies to the specified environment.

Resource scopes



When designing your deployment processes, it is important to understand the scope of all the resources and their dependencies.

Some Apigee objects, such as users roles, companies, and API products, are scoped at the organization level,

But other objects, such as target servers, KVMs, and cache resources, are scoped at the environment level.

Import/export API proxies

Export API proxy bundle:

```
curl -u <sysAdminEmail>:<passwd>
http://<ms_IP>:<port>/v1/o/<orgname>/apis/<apiname>/revisions/<revision>?
format=bundle -o <apiname>.zip
```

Import API proxy bundle:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H "Content-Type:
application/octet-stream" -T <apiproxybundle.zip>
"http://<ms_IP>:<port>/v1/o/<orgname>/apis?action=import&name=<apiname>"
```

You can use management APIs to export and import API proxies and configuration resources.

To export an API proxy, use the /apis management API and provide the organization name, api name, and revision of the proxy. You must also supply the format query parameter in the request.

If the call is successful, the proxy bundle will be saved to your local machine as a zip file containing XML descriptions of the proxy, policies, targets, and associated resource files.

To import a proxy, use a POST call to the same API by setting the content type header to application/octet-stream and provide the name of the zip file to be imported.

You must also supply the action and name of the API as query parameters in the request.

If you import a proxy that already exists in the organization, it is automatically saved with a new revision number.

Agenda

Platform operation

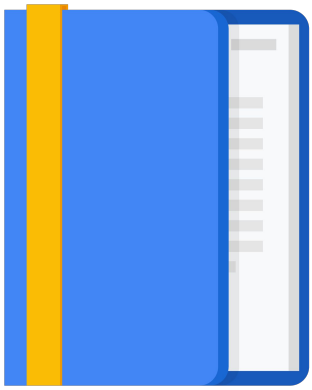
Infrastructure operation

REST fundamentals

Management API

API proxy deployment

Edge analytics



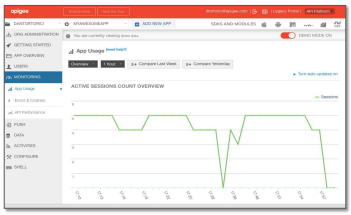
In this lesson we'll provide an overview of analytics in Apigee Edge.

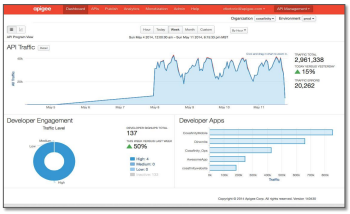
We discuss the analytics data flow, reporting, and analytics groups, and how to manage them.

Analytics

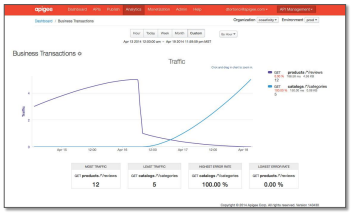
Operational Metrics

App Performance Monitoring





Developer Metrics



Business Metrics

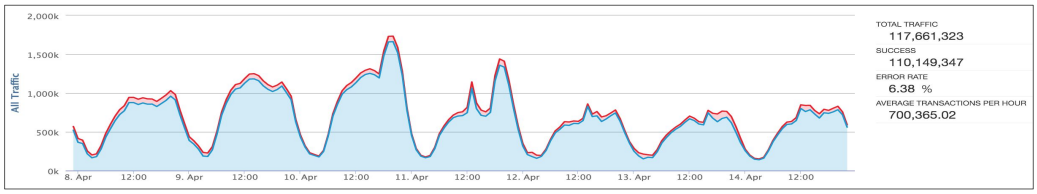
Edge Analytics Services enables end-to-end visibility of the Apigee private cloud platform with unified operational, developer, application performance, and business metrics that are required to monitor, measure, and manage your API program.

Analytics Services offers dashboards and reports that use custom variables, dimensions, drill-downs, metric correlations, and filters.

Analytics reporting

Analytics reporting explained:
https://docs.apigee.com/api-platform/analytics/using-analytics-dashboards

Analytics API:
https://docs.apigee.com/api-platform/analytics/use-analytics-api-measure-api-program-performance



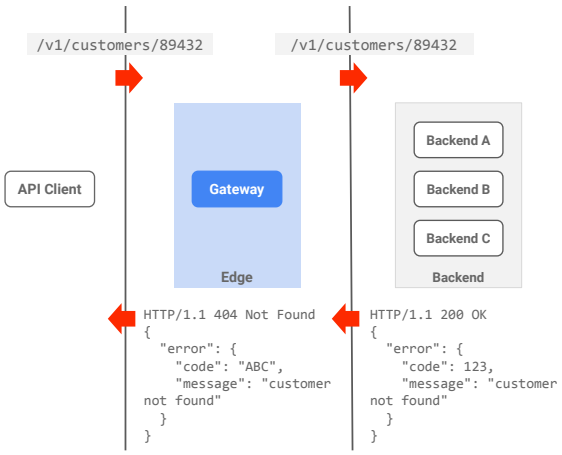
For detailed information about analytics reporting and the analytics API, the Apigee documentation website has several resources at the links provided.

Various reporting dashboards are available that plot API proxy performance, application transactions, area code analysis, and traffic composition.

The documentation also covers all aspects of the Analytics API and how to use it to measure API performance.

API specification and backend errors

- Edge analytics reports present the Proxy view and the Target view.
- Errors in backend systems may not match the same error specification implemented by your API proxies running on Edge.
- Operations teams must be aware of this error mapping in the API when troubleshooting issues.



Before reviewing the capabilities of Apigee Edge Analytics, let's talk about API and backend errors.

It is important to remember that Edge Analytics include both the proxy and the target report views.

In the example, the presentation of the error by the proxy (404) is different from that of the backend target server (200).

The backend system returns a 200 OK response with a payload containing the error message and an error code.

This is an error condition, so the API proxy converts that error and returns the correct 404 Not Found HTTP status code and message to the API client.

Understanding this error mapping between the API and the backend systems is important for operations teams to troubleshoot issues with the platform.

Analytics data partitioning

- Analytics data is partitioned by organization and environment.
- Raw data is aggregated, and the aggregated data is presented in the Enterprise UI.
- Analytics data can also be queried using the management API.


```
# psql -h /opt/apigee/var/run/apigee-postgresql -U apigee apigee
apigee=# \d analytics."<orgname>.<envname>.fact"
apigee=# \q
```

Let's now discuss the analytics capabilities provided in Apigee Edge for private cloud.

Edge analytics data is stored in a PostgreSQL database. The stored data is partitioned by both organization and environment.

The database schema includes several tables that use the naming convention: organizationname.environmentname.fact.

We will see in a later section how this partitioning allows for easy scaling out of the Analytics components.

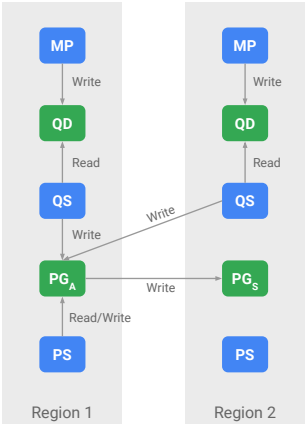
To increase the performance of report generation in the Enterprise UI, raw data is aggregated by a separate Apigee service.

This aggregated data is presented as reports and charts in the Enterprise UI.

You can also query analytics data using the management API.

Analytics data flow

- Analytics data is generated by the Message Processor and asynchronously send to the Qpid daemon.
- The Qpid server consumes raw analytics data and stores it in the PostgreSQL database.
- Data in PostgreSQL is logically partitioned by organization and environment.
- The Postgres server aggregates the raw data.
- The analytics record size is approx. 2 kb. It will vary if custom variables are captured.
- Analytics components are logically assigned to organizations and environments using analytics groups.



It is important to understand the Analytics data flow.

Analytics data is generated on the message processor and sent asynchronously to the Qpid daemon.

The Qpid server component consumes the raw analytics data from the Qpid daemon and writes it to the PostgreSQL database.

As previously mentioned, the data is partitioned logically by organization and environment.

A separate Edge component, the Postgres server, aggregates the data stored in PostgreSQL.

When sizing Postgres machines, it's important to know that the average analytics record size is around 2kb. If custom variables are used, this record size will increase.

The Qpid daemon, Qpid server, PostgreSQL and Postgres server components are logically assigned to organizations and environments using analytics groups.

Writing analytics data to a file

- By default, analytics data collected by the Message Processor is uploaded to Qpid and PostgreSQL for processing.
- Alternatively, you can configure the Message Processor to write analytics data to disk.
- You can also use both options.
- If you enable this option, you are responsible for file rotation.

Full documentation of the process can be found here:
https://docs.apigee.com/private-cloud/latest/write-analytics-file

By default, analytics data collected by the Message Processor is uploaded to Qpid and PostgreSQL for processing. You can then view the analytics data in the Edge UI.

Alternatively, you can configure the Message Processor to write analytics data to disk. You can then upload that data to your own analytics system for analysis.

For example, you could upload the data to BigQuery. You can then take advantage of the powerful query and machine learning capabilities offered by BigQuery and TensorFlow to perform your own data analysis.

Note that If you choose to only write to a file on the Message processor, the analytics data would not populate to the Postgres server and the Enterprise UI dashboards will be empty.

Alternatively, you can choose to use both options.

Remember that, when writing analytics data to a file, you must manage file rotation and clean up.

The process of enabling and managing local analytics files is documented at the link provided.

Analytics groups

Analytics groups (axgroup) are data structures that describe the Qpid and Postgres servers associated to a given organization and environment.

```
curl -u <sysAdminEmail>:<passwd> "http://<ms_IP>:8080/v1/analytics/groups/ax"
[ {
  "name" : "axgroup001",
  "properties" : {
    "consumer-type" : "ax"
  },
  "scopes" : [ "traininglab~prod" ],
  "uuids" : {
    "postgres-server" : [ "baf7d9d5-618c-4ce4-8c5b-23ad2c62eb51:489aa9b0-b764-4ade-8615-27db09e9deca" ],
    "qpid-server" : [ "708cf21f-2efe-41c9-97b5-809e56676347", "4f1c20e5-0dc5-450c-8619-3cd9fe75ce4b" ]
  },
  "consumer-groups" : [ {
    "name" : "consumer-group-001",
    "consumers" : [ "708cf21f-2efe-41c9-97b5-809e56676347", "4f1c20e5-0dc5-450c-8619-3cd9fe75ce4b" ],
    "datastores" : [ "baf7d9d5-618c-4ce4-8c5b-23ad2c62eb51:489aa9b0-b764-4ade-8615-27db09e9deca" ],
    "properties" : {
    }
  } ],
  "data-processors" : {
  }
} ]
```

Analytics Groups are data structures that describe the Qpid and Postgres servers associated to any given organization and environment.

The example shows the response from the analytics groups management API that includes all of the available analytics groups.

In this case, the response contains a single group named axgroup001 and a single scope named traininglab~prod. This corresponds to an environment name prod in the training lab organization.

Next, the unique uuids for the Postgres and qpid-servers are listed.

Finally, consumer groups are listed that contain the qpid server uuids as consumers and the postgres server uuids as datastores.

Creating an AX group

The following API calls allow you to create a new analytics group:

```
curl –v -u <sysAdminEmail>:<passwd> -X POST -H "Content-Type: application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>"

curl –v -u <sysAdminEmail>:<passwd> -X POST –H "Content-Type:application/json" –H
"Accept:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/consumer-groups?name=consume
r-group-001"

curl –v -u <sysAdminEmail>:<passwd> -X POST –H "Content-Type:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/properties?propName=consumer
-type&propValue=ax"

curl –v -u <sysAdminEmail>:<passwd> -X POST –H "Content-Type:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/properties?propName=region&p
ropValue=<region>"
```

Most customers will never need to create new Analytics Groups.

The exception to this is if you are manually configuring groups, which is uncommon, or horizontally scaling the analytics pipeline.

Analytics groups are created by using management API calls.

The general process to follow is to first create a new analytics group and then create a new consumer group within that analytics group.

You then set the consumer type and region properties for the newly created analytics group.

Add Postgres servers to a new axgroup

Add Postgres server:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H "Content-Type: application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/servers?uuid=<activeu
uid>,<standbyuuid>&type=postgres-server&force=true"
```

Add data store:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H "Accept:application/json"
-H"Content-Type:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/consumer-groups/<cons
umer-group>/datastores?uuid=<activeuuid>,<standbyuuid>"
```

After the new analytics group is created, you populate it with postgres and qpid servers.

This is done using the analytics groups management API calls.

To add the servers to the group, use a POST request to the server's endpoint.

To add the datastore, use a POST request to the consumer-group datastores endpoint.

Add Qpid servers to a new axgroup

Add qpid server:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H "Content-Type: application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/servers?uuid=<qpiduui
d>&type=qpid-server"
```

Add to consumer groups:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H"Content-Type:application/json" -H
"Accept:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/consumer-groups/<cons
umer-group>/consumers?uuid=<qpiduuid>"
```

A similar process is followed to add the Qpid servers to the analytics and consumer groups.

Multiple qpid servers can be added one at a time.

Note that the qpid-server must be stopped before the analytics group is updated, and re-started when the update is complete.

Updating axgroup

Removing the existing Postgres UUID:

```
curl -v -u <sysAdminEmail>:<passwd> -X DELETE –H "Accept:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/consumer-groups/<cons
umer-group>/datastores/<activeuuid>,<standbyuuid>"


curl -v -u <sysAdminEmail>:<passwd> -X DELETE –H "Accept:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/servers?uuid=<activeu
uid>,<standbyuuid>&type=postgres-server"
```

In order to update the postgres server entries in the analytics group, you must first delete the existing postgres server UUIDs from the group.

This is accomplished by making a DELETE API call to the analytics groups servers and consumer groups datastore endpoints.

You must supply the UUIDs of the postgres servers being removed.

Updating axgroup

Adding new Postgres UUIDs:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H "Accept:application/json"
-H "Content-Type:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/servers?uuid=<a
ctiveuuid>,<standbyuuid>&type=postgres-server"
```

```
curl -v -u <sysAdminEmail>:<passwd> -X POST -H "Accept:application/json"
-H "Content-Type:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/consumer-groups
/<consumer-group>/datastores?uuid=<activeuuid>,<standbyuuid>"
```

You can then add new postgres server UUIDs to the analytics group by using POST calls to the same analytics groups management API as discussed earlier.

Deleting from axgroup

Delete a Qpid server from axgroup:

```
curl -v -u <sysAdminEmail>:<passwd> -X DELETE -H "Accept:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/consumer-groups
/<consumer-group>/consumers/<qpiduuid>"

curl -v -u <sysAdminEmail>:<passwd> -X DELETE -H "Accept:application/json"
"http://<ms_IP>:8080/v1/analytics/groups/ax/<axgroup-name>/servers?uuid=<q
piduuid>&type=qpid-server”
```

To remove a qpid server from an analytics group, you make a DELETE API call to the analytics groups servers and consumer-group consumers endpoints.

You must supply the UUID of the qpid server that is being removed.

Move scope from one axgroup to another axgroup

Add the scope to new axgroup:

```
curl -v -u <sysAdminEmail>:<passwd> -X POST
"http://<ms_IP>:8080/v1/analytics/groups/ax/<new-axgroup-name>/scopes?org=
<org>&env=<env>" -H "content-type: application/json"
```

To confirm the scope addition:

```
curl -v -u <sysAdminEmail>:<passwd> -X GET
"http://<ms_IP>:8080/v1/analytics/groups/ax/<new-axgroup-name>/"
```

Remove the scope from the old axgroup:

```
curl -v -u <sysAdminEmail>:<passwd> -X DELETE
"http://<ms_IP>:8080/v1/analytics/groups/ax/<old-axgroup-name>/scopes?org=
<org>&env=<env>"
```

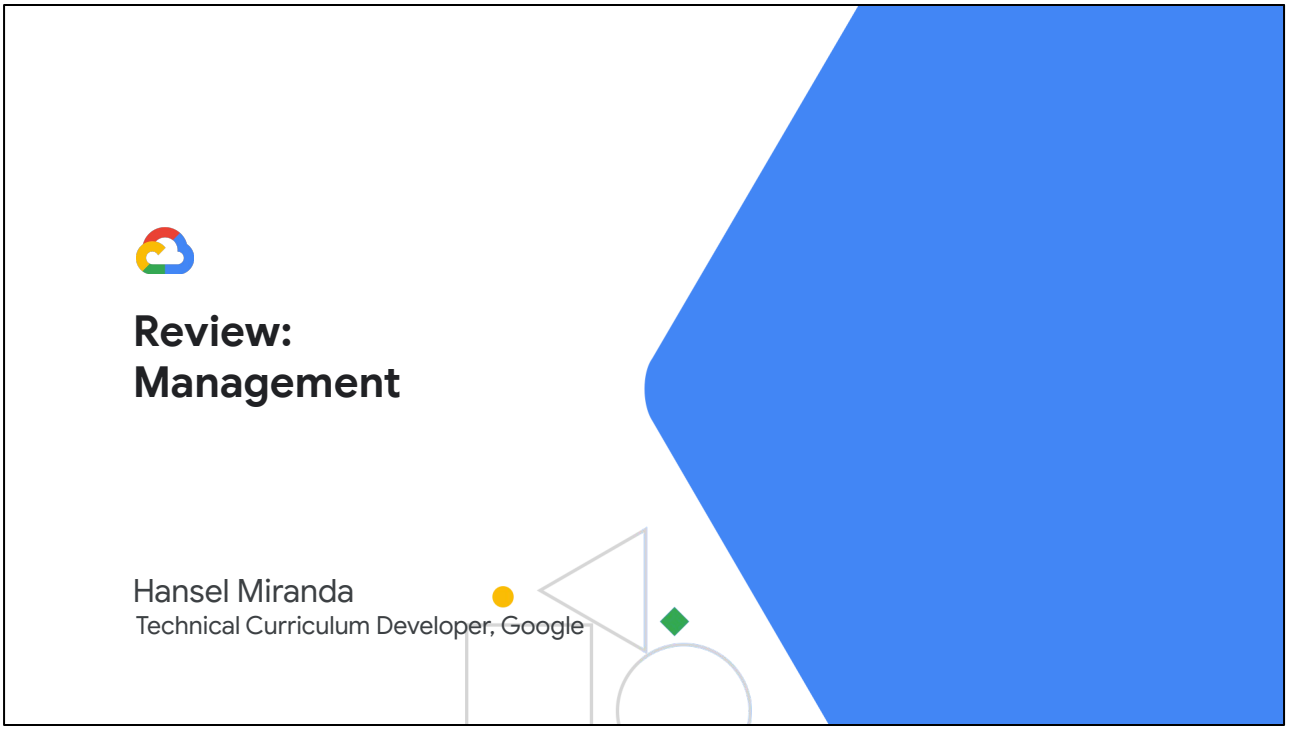
When horizontally scaling analytics groups, you may want to move one or more environments to a new analytics group.

This is done by first creating the new analytics group.

Next, the scope representing your organization and environment name is added to the new group by using an API call.

After this is verified, use another API call to delete the scope from the current analytics group.

Note that this action requires a rolling restart of all the message processors that serve traffic for that particular scope.



In this module we discussed various aspects of the platform and infrastructure operation of the Apigee Edge private cloud platform.

This included configuring Apigee Edge, user management and maintenance operations like backup and restore.

We discussed REST API concepts and provided an overview of the Apigee management API.

You also learned about API proxy deployment and how you can use Apigee analytics to track the operational performance of the platform.


Google Cloud