Users are able to deploy, monitor, and configure their Meraki devices via the Meraki dashboard web interface or via APIs. Once a user makes a configuration change, the change request is sent to the Meraki cloud and is then pushed to the relevant device(s).

**The Meraki dashboard:**A modern web browser-based tool used to configure Meraki devices and services.

**Management data:**The data (configuration, statistics, monitoring, etc.) that **flows from Meraki devices** (wireless access points, switches, security appliances) **to the Meraki cloud** over a **secure internet connection**.

**User data:**Data related to user traffic (web browsing, internal applications, etc.). **User data does not flow through the Meraki cloud**, instead flowing directly to their destination on the LAN or across the WAN.

The Meraki cloud is the backbone of the Meraki management solution. This "cloud" is a collection of highly reliable multi-tenant servers strategically distributed around the world at Meraki data centers. The servers at these data centers are powerful hosting computers.

**All Meraki services (the dashboard and APIs)** are also replicated across multiple independent data centers, so they can failover rapidly in the event of a catastrophic data center failure.

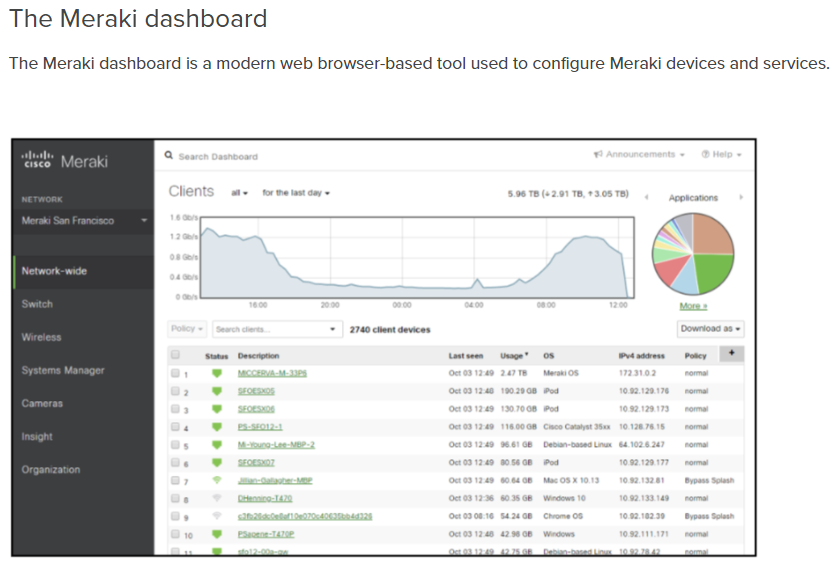
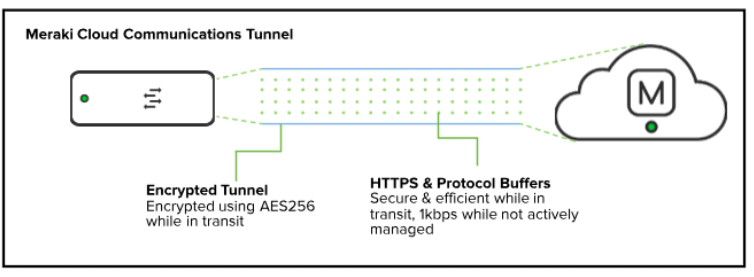
**Meraki Device-to-Cloud Communications**

Meraki uses an event-driven remote procedure call (RPC) engine for Meraki devices to communicate to the dashboard and for Meraki servers to send and receive data. Meraki hardware devices act as the server/receiver as the Meraki cloud initiates calls to the devices for data collection and configuration deployment. The cloud infrastructure is the initiator, so configurations can be executed in the cloud before the devices are actually online or even physically deployed.

As the device runs on the network, it will communicate device and network usage analytics back to the Meraki cloud. **Dashboard analytics based on this information, in the form of graphs and charts**, are updated regularly in the Meraki cloud and are displayed in the dashboard of users when they are viewing this information.

### Meraki Device-to-Cloud Communications

Meraki uses an event-driven remote procedure call (RPC) engine for Meraki devices to communicate to the dashboard and for Meraki servers to send and receive data. Meraki hardware devices act as the server/receiver as the Meraki cloud initiates calls to the devices for data collection and configuration deployment. The cloud infrastructure is the initiator, so configurations can be executed in the cloud before the devices are actually online or even physically deployed.



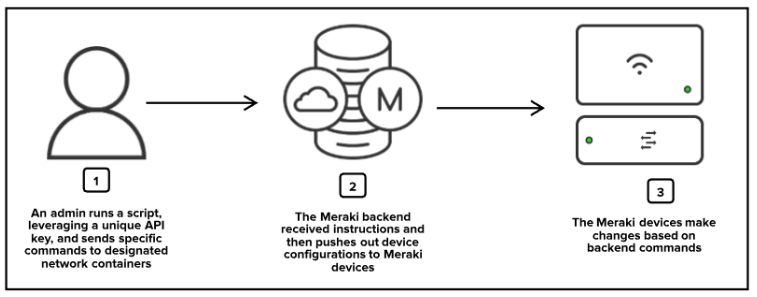
The Meraki dashboard **is the visual alternative to the traditional command line**, which is used to manage many routers, switches, security devices, and more. Instead, Meraki puts all devices within networks in one place and allows users to apply changes in a simple, easy-to-use format.

In addition to simplifying device management, **the dashboard is also a platform for viewing network analytics, applying network permissions, and keeping track of users**. The dashboard allows users to view camera streams, manage users’ mobile devices and computers, set content rules, and monitor upstream connections from a single place.

##### Meraki APIs

Meraki APIs provide control of the Meraki solution in a programmable way, enabling actions that may not be possible with the dashboard, or proving more granular control. Meraki APIs are RESTful APIs using HTTPS for transport and JSON for object serialization.

By providing open API accessibility, Meraki leverages the power of the cloud platform on a deeper level to create more efficient and powerful solutions. Through Meraki APIs, users can automate deployments, monitor their networks, and build additional solutions on top of the Meraki dashboard.



Additionally, it is important to consider **Meraki server and data center limits**. Meraki server architecture is a multi-tenant solution that hosts multiple customers on the same hardware with secure permissions-based segmentation among them. The maximum scale supported in a single organization is 25,000 physical Meraki devices. *If a single business intends to have* more than 25,000 Meraki devices in their solution, they are strongly encouraged to work with their account team to design a deployment strategy across multiple organizations.

##### Administrators

There are two basic types of dashboard administrators: **Organization administrators** and **Network administrators**.

* **Organization administrators (Organization ID)** have complete access to their organization and all its networks. This type of account is equivalent to a root or domain admin, so it is important to carefully maintain who has this level of control.
  + **Organization - Read-only**: The user is able to access/view most aspects of network and organization-wide settings, but is unable to make any changes.
  + **Organization - Full**: The user has full administrative access to all networks and organization-wide settings. This is the highest level of access available.
* **Network(Group) administrators – Group ID** have access to individual networks and their devices. These users can have complete or limited control over their network configuration, but do not have access to organization-level information (licensing, device inventory, etc).
  + **Network - Guest ambassador**: The user is only able to see the list of Meraki authentication users, add users, update existing users, and authorize/de-authorize users on an SSID or Client VPN. Ambassadors can also remove wireless users, if they are an ambassador on all networks.
  + **Network - Monitor-only**: The user is only able to view a subset of the **Monitor** section in the dashboard and no changes can be made. This can be useful for proving networking monitoring access to customers in service provider deployments.
  + **Network - Read-only**: The user is able to access most aspects of a network, including the **Configure** section of the dashboard, but no changes can be made.
  + **Network - Full**: The user has access to view all aspects of a network and make any changes to it.

SDKs

Going forward, the custom Meraki [Python library](https://developer.cisco.com/meraki/api-v1/#python) will be the recommended SDK for simplified API scripting. The previously auto-generated Python, Node.js, and Ruby SDKs for v0 will remain in the Meraki GitHub but will no longer be maintained

# Global Meraki Path Parameter IDs

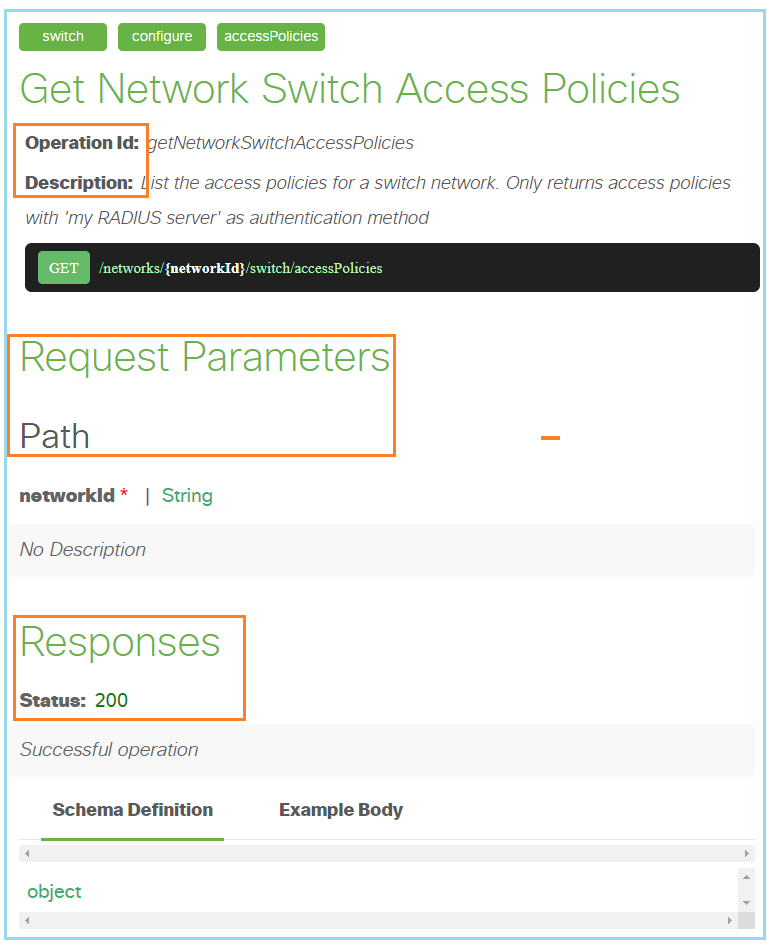
# <https://developer.cisco.com/meraki/api-v1/#!path-parameters/global-meraki-path-parameter-ids>

Each service API path parameter ID has related API endpoint operations which can be used to get the ID's value.

For example, the following API path requires an organizationId for its scope and an adminId for its service parameter.

**/organizations/{organizationId}/admins/{adminId}**

Some endpoints will return the parameter named as id in the body, whereas other operations may return the named ID such as groupId or floorPlanId. This table provides a helpful map of the global Meraki path parameters and their respective operations. It also includes the ID parameter name you should expect to use for related API operations.



**Meraki Dashboard API Python Library**

The Meraki Dashboard API [Python library](https://github.com/meraki/dashboard-api-python/) provides all current Meraki [Dashboard API](https://developer.cisco.com/docs/meraki-api-v1) calls to interface with the Cisco Meraki cloud-managed platform. The library is supported on Python 3.7 or above, and you can install it via [PyPI](https://pypi.org/project/meraki/):

CODE SNIPPET

pip install meraki

Features

While you can make direct HTTP requests to dashboard API in any programming language or REST API client, using a client library can make it easier for you to focus on your specific use case, without the overhead of having to write functions to handle the dashboard API calls. The Python library can also take care of error handling, logging, retries, and other convenient processes and options for you automatically.

* Support for all API endpoints, as it uses the [OpenAPI specification](https://api.meraki.com/api/v1/openapiSpec) to generate source code
* Log all API requests made to a local file as well as on-screen console
* Automatic retries upon 429 rate limit errors, using the [Retry-After field](https://developer.cisco.com/docs/meraki-api-v1/#!rate-limit) within response headers
* Get all (or a specified number of) pages of data with built-in pagination control
* Tweak settings such as maximum retries, certificate path, suppress logging, and other options
* Simulate POST/PUT/DELETE calls to preview first, so that network configuration does not get changed
* Includes the legacy module's (version 0.34 and prior) functions for backward compatibility

## **Setup**

1. Enable API access in your Meraki dashboard organization and obtain an API key ([instructions](https://documentation.meraki.com/zGeneral_Administration/Other_Topics/The_Cisco_Meraki_Dashboard_API))
2. **Keep your API key safe and secure, as it is similar to a password for your dashboard**. If publishing your Python code to a wider audience, please research secure handling of API keys.
3. Install the latest version of [Python 3](https://wiki.python.org/moin/BeginnersGuide/NonProgrammers)
4. Use pip (or an alternative such as easy\_install) to install the library from the Python [Package Index](https://pypi.org/project/meraki/):
   * **pip install meraki**
   * If you have both Python3 and Python2 installed, you may need to use pip3 (so pip3 install meraki) along with python3 on your system
   * If meraki was previously installed, you can upgrade to the latest non-beta release with **pip install --upgrade meraki**
5. If you clone this repository and want to use v1 locally, rename the folder "meraki\_v1" to "meraki", replacing the v0 contents there. You can also specify the version of the library when installing with pip:
   * See the full [release history](https://pypi.org/project/meraki/#history) to pick the version you want, or use pip install meraki== without including a version number to display the list of available versions
   * v0 versions of the Python library begin with 0 (0.**x**.**y**), and v1 versions begin with 1 (1.0.0b**z** for beta)
   * Specify the version you want with the install command; for example: pip install meraki==0.x.y for v0 or pip install meraki==1.0.0bz for v1 beta
   * You can also see the version currently installed with pip show meraki

## **Usage**

1. Export your API key as an [environment variable](https://www.twilio.com/blog/2017/01/how-to-set-environment-variables.html), for example:

SHELL

export MERAKI\_DASHBOARD\_API\_KEY=093b24e85df15a3e66f1fc359f4c48493eaa1b73

1. Alternatively, define your API key as a variable in your source code; this method is not recommended due to its inherent insecurity.
2. Single line of code to import and use the library goes at the top of your script:

import meraki

1. Instantiate the client (API consumer class), optionally specifying any of the parameters available to set:

dashboard = meraki.DashboardAPI()

1. Make dashboard API calls in your source code, using the format client.scope.operation, where client is the name you defined in the previous step (**dashboard** above), scope is the corresponding scope that represents the first tag from the OpenAPI spec, and operation is the operation of the API endpoint. For example, to make a call to get the list of organizations accessible by the API key defined in step 1, use this function call:

my\_orgs = dashboard.organizations.getOrganizations()

1. If you were using this module versions 0.34 and prior, that file's functions are included in the legacy.py file, and you can adapt your existing scripts by replacing their from meraki import meraki line to **import meraki**

[**https://developer.cisco.com/meraki/api-v1/#!update-network**](https://developer.cisco.com/meraki/api-v1/#!update-network)

[**https://developer.cisco.com/meraki/api-v1/#!getting-started/base-uri**](https://developer.cisco.com/meraki/api-v1/#!getting-started/base-uri)

For a full working script that demos this library, please see and run the **org\_wide\_clients\_v1.py** file included (in **examples** folder). That code collects the clients of all networks, in all orgs to which the key has access. No changes are made, since only GET endpoints are called, and the data is written to local CSV output files.