Cisco Meraki Guide

What is Cisco Meraki?

Cisco Meraki is an easy-to-use, cloud-based, network infrastructure platform for enterprise environments. While most network hardware uses command-line interfaces (CLIs) for configuration, Meraki uses an easy-to-use Dashboard hosted in the Meraki cloud. No on-premises management hardware or software is required - only the network infrastructure to run your business.

MS Switches

Meraki MS switches come in multiple flavors and form factors. Meraki switches support 10/100/1000/10000 ports, as well as Cisco's mGig technology for 2.5/5/10Gbps copper connectivity. 8, 24, and 48 port flavors are available with PoE (802.3af/802.3at/UPoE) available on many models.

MX Firewalls

Meraki's MX firewalls support full layer 3-7 deep packet inspection. MX firewalls are compatible with a variety of VPN technologies including IPSec, SSL VPN, and Meraki's easy-to-use AutoVPN.

MR Wireless Access Points

MR access points are enterprise-class, high-performance access points for the enterprise. MR access points have MIMO technology and integrated beamforming built-in for high performance applications. BLE allows for advanced location applications to be developed with no on-premises analytics platforms.

Using the Meraki modules

Meraki modules provide a user-friendly interface to manage your Meraki environment using Ansible. For example, details about SNMP settings for a particular organization can be discovered using the module *meraki_snmp <meraki_snmp_module>*.

```
- name: Query SNMP settings
  meraki_snmp:
    api_key: abc123
    org_name: AcmeCorp
    state: query
  delegate_to: localhost
```

Information about a particular object can be queried. For example, the *meraki_admin <meraki_admin_module>* module supports

```
- name: Gather information about Jane Doe
  meraki_admin:
    api_key: abc123
    org_name: AcmeCorp
    state: query
    email: janedoe@email.com
  delegate_to: localhost
```

Common Parameters

All Ansible Meraki modules support the following parameters which affect communication with the Meraki Dashboard API. Most of these should only be used by Meraki developers and not the general public.

host
: Hostname or IP of Meraki Dashboard.

use_https
: Specifies whether communication should be over HTTPS. (Defaults to `yes`)

use_proxy
: Whether to use a proxy for any communication.

validate_certs
: Determine whether certificates should be validated or trusted. (Defaults to `yes`)

These are the common parameters which are used for most every module.

org_name
: Name of organization to perform actions in.

org_id
: ID of organization to perform actions in.

net_name
: Name of network to perform actions in.

net_id
: ID of network to perform actions in.

state
: General specification of what action to take. `query` does lookups. `present` creates or edits. `absent` deletes.

> **Hint**
>
> Use the `org_id` and `net_id` parameters when possible. `org_name` and `net_name` require additional behind-the-scenes API calls to learn the ID values. `org_id` and `net_id` will perform faster.

Meraki Authentication

All API access with the Meraki Dashboard requires an API key. An API key can be generated from the organization's settings page. Each play in a playbook requires the `api_key` parameter to be specified.

The "Vault" feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See :ref:`playbooks_vault` for more information.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\scenario_guides\[ansible-devel][docs][docsite][rst][scenario_guides]guide_meraki.rst`, line 100); *backlink***
>
> Unknown interpreted text role "ref".

Meraki's API returns a 404 error if the API key is not correct. It does not provide any specific error saying the key is incorrect. If you receive a 404 error, check the API key first.

Returned Data Structures

Meraki and its related Ansible modules return most information in the form of a list. For example, this is returned information by `meraki_admin` querying administrators. It returns a list even though there's only one.

```
[
    {
        "orgAccess": "full",
        "name": "John Doe",
        "tags": [],
        "networks": [],
        "email": "john@doe.com",
        "id": "12345677890"
    }
]
```

Handling Returned Data

Since Meraki's response data uses lists instead of properly keyed dictionaries for responses, certain strategies should be used when querying data for particular information. For many situations, use the `selectattr()` Jinja2 function.

Merging Existing and New Data

Ansible's Meraki modules do not allow for manipulating data. For example, you may need to insert a rule in the middle of a firewall ruleset. Ansible and the Meraki modules lack a way to directly merge to manipulate data. However, a playlist can use a few tasks to split the list where you need to insert a rule and then merge them together again with the new rule added. The steps involved are as follows:

1. Create blank "front" and "back" lists.

```
vars:
  - front_rules: []
  - back_rules: []
```

2. Get existing firewall rules from Meraki and create a new variable.

```
- name: Get firewall rules
  meraki_mx_l3_firewall:
    auth_key: abc123
    org_name: YourOrg
    net_name: YourNet
    state: query
  delegate_to: localhost
  register: rules
- set_fact:
    original_ruleset: '{{rules.data}}'
```

3. Write the new rule. The new rule needs to be in a list so it can be merged with other lists in an upcoming step. The blank - puts the rule in a list so it can be merged.

```
- set_fact:
    new_rule:
      -
        - comment: Block traffic to server
          src_cidr: 192.0.1.0/24
          src_port: any
          dst_cidr: 192.0.1.2/32
          dst_port: any
          protocol: any
          policy: deny
```

4. Split the rules into two lists. This assumes the existing ruleset is 2 rules long.

```
- set_fact:
    front_rules: '{{front_rules + [ original_ruleset[:1] ]}}'
- set_fact:
    back_rules: '{{back_rules + [ original_ruleset[1:] ]}}'
```

5. Merge rules with the new rule in the middle.

```
- set_fact:
    new_ruleset: '{{front_rules + new_rule + back_rules}}'
```

6. Upload new ruleset to Meraki.

```
- name: Set two firewall rules
  meraki_mx_l3_firewall:
    auth_key: abc123
    org_name: YourOrg
    net_name: YourNet
    state: present
    rules: '{{ new_ruleset }}'
  delegate_to: localhost
```

Error Handling

Ansible's Meraki modules will often fail if improper or incompatible parameters are specified. However, there will likely be scenarios where the module accepts the information but the Meraki API rejects the data. If this happens, the error will be returned in the `body` field for HTTP status of 400 return code.

Meraki's API returns a 404 error if the API key is not correct. It does not provide any specific error saying the key is incorrect. If you receive a 404 error, check the API key first. 404 errors can also occur if improper object IDs (ex. `org_id`) are specified.