DevNet Associate v1.0

Cisco Network Management

8.4.1

Network Management Platforms

Cisco has a long history of inventing and supporting networking solutions, in both hardware and software. Since the beginning of computer networking, network configuration centered on a device-by-device, manual, and often complex set of tasks. This approach did not pose much of a problem with a few networking devices, but configuring and maintaining hundreds or even thousands of devices on a network becomes nearly impossible.

Moreover, as scale increases, changes implemented by humans have a higher chance of misconfigurations. The mistakes could be simple typos, applying a new change to the wrong device, or even completely missing a device. Manually performing repetitive tasks that demand a high degree of consistency introduces a risk for error. And the number of changes required increases as there are more demands from the business to deploy more applications at a faster rate.

You probably know that the solution lies in automation. Network programmability helps reduce Operational Expenses (OPEX), which represents a significant portion of overall network cost. Teams can speed up service delivery by automating tasks that are typically done by hand using a command-line interface (CLI).

Network automation plays a crucial part in simplifying day-to-day operations and maintenance. When you automate everyday network tasks and functions, and manage repetitive processes with code, you can reduce human errors and improve network service availability. Operations teams can respond and handle trouble tickets faster, and can even act proactively. Network automation also lowers costs by giving operations teams the ability to migrate mundane and repetitive tasks to automated processes.

Network automation is used for various common tasks in an organization:

- **Device provisioning -** Device provisioning is simply configuring network devices more efficiently, faster, and with fewer errors because human interaction with each network device is decreased.
- **Device software management -** Controlling the download and deployment of software updates is a relatively simple task, but it can be time-consuming and may fall prey to human error. Many automated tools have been created to address this issue, but they can lag behind customer requirements. A simple network programmability solution for device software management is beneficial in many environments.
- **Compliance checks -** Network automation methods allow the unique ability to quickly audit large groups of network devices for configuration errors and automatically make the appropriate corrections

with built-in regression tests.
- **Reporting -** Automation decreases the manual effort that is needed to extract information and coordinate data from disparate information sources to create meaningful and human-readable reports.
- **Troubleshooting -** Network automation makes troubleshooting easier by making configuration analysis and real-time error checking fast and simple, even with many network devices.
- **Data collection and telemetry -** A common part of effectively maintaining a network is collecting data from network devices and telemetry on network behavior. Even the way data is collected is changing as now many devices can push data (and stream) off box, in real-time, in contrast to being polled in regular time intervals.

Take a look at the products that Cisco provides for all of these common tasks, especially those with an eye for automation.

8.4.2

Cisco IOS XE

IOS stands for "Internetwork Operating System." IOS was the original operating system for Cisco Systems routers and Cisco network switches. IOS XE is the next-generation programmable platform. With IOS XE, you have access to standards-based, consistent, programmable interfaces, standard data models for configuration, deployment, and rollback, as well as services integration with the network.

Benefits and purpose

IOS XE is based on Linux. Key network functions run as system daemons and system processes, which results in greater stability. Distributing functions into processes also allows for better workload balance, resulting in higher performance.

In IOS XE the control plane and the data plane are separated. The control plane "controls" the network, meaning it stores the routes, mapping, generally all the "signals" that are needed to run the router or switch. The data plane contains the actual client, user, or application data that needs to go along the network from one device to another.

The IOS XE system includes configuration management, provisioning, and diagnostic services. Tracing capabilities, also included, support quick resolution of network problems. With Model-driven programmability included, you can create and extend automated device management.

Model-driven programmability support in Cisco IOS XE

With YANG Data Models available for each Cisco device, you have access to configuration data, state information, and any data specific to the exact Cisco device model. Cisco device YANG models can be obtained from the Cisco Device YANG models repository.

IOS XE is used on most high-end Cisco platforms, including switches, routers, gateways. Not all programmability features are available on every platform. Consult the reference documentation for your device and release version. See Networking Software (IOS & NX-OS) and Programmability Configuration Guide.

NETCONF/YANG is supported as of IOS XE 16.3.1 software. In Cisco IOS XE Fuji 16.8.1 and later releases, operational data works on platforms running NETCONF and is enabled by default.

Enabling model-driven programmability

On some devices, Model-driven programmability services must first be enabled.

Enter configuration mode using the `configure terminal` command. (See configure terminal reference for details.)

Enable NETCONF on IOS XE

NETCONF connections should be authenticated using AAA credentials. Typically RADIUS, TACACS+, or local users defined with privilege level 15 access are allowed, but running from a particular privilege mode varies depending on the platform.

The CLI command to enable NETCONF is:

```
CSR1kv> enable
CSR1kv# configure terminal
CSR1kv(config)# netconf-yang
```

Troubleshooting

If you get an "Unknown command" response to the `netconf-yang` as shown below, double-check the device configuration.

```
CSR1kv(config)# netconf-yang
% Bad IP address or host name% Unknown command or computer name, or unable to find
computer address
```

Cisco IOS XE supports two datastores: running and candidate. It also supports locking datastores, as well as configuration rollback.

Enable RESTCONF on IOS XE

RESTCONF connections should be authenticated using AAA credentials. RADIUS, TACACS+, or local users defined with privilege level 15 access are allowed.

RESTCONF runs over HTTPS. The following commands must be enabled to support RESTCONF over port 443:

```
CSR1kv(config)# ip http secure-server
```

The CLI command to enable RESTCONF is:

```
CSR1kv(config)# restconf
```

When enabled using the CLI, all supported operations may be governed through model interfaces, including optional settings for RESTCONF configuration and operational data settings.

Accessing YANG models

For a complete look at Cisco YANG models, browse or clone the GitHub repository at https://github.com/YangModels/yang. The `vendor/cisco` subdirectory contains models for Cisco platforms.

8.4.3

Cisco DNA Center

A Cisco DNA Center is a foundational controller and analytics platform for large and midsize organizations. It is at the heart of a Cisco intent-based network. It provides a single dashboard for network management, network automation, network assurance, monitoring, analytics, and security.

Cisco DNA Center provides both a web-based GUI dashboard and the RESTful Intent API used to programmatically access its services and functions.

It supports full 360-degree services and integration:

- **North** - The Intent API provides specific capabilities of the Cisco DNA Center platform.
- **East** - These are services for asynchronous Event Notification through WebHooks and email notification.
- **Southbound** - The Multivendor SDK is used to integrate non-Cisco devices into the Cisco DNA network.
- **Westbound** - Integration with Assurance and IT Service Management (ITSM) services, such as ServiceNow.

Here the focus is on using the Intent API, but you will also get a quick overview of services directly available through the GUI.

Cisco DNA Center dashboard (GUI)

The GUI organizes services and activities into Design, Policy, Provision, Assurance and Platform.

- **Design** - Design your network using intuitive workflows, starting with locations where your network devices will be deployed. Existing network designs created in Cisco Prime Infrastructure or Application Policy Infrastructure can be imported into Cisco DNA Center.
- **Policy** - Define and manage user and device profiles to facilitate highly secure access and network segmentation.
- **Provision** - Deployment, and Policy-based automation to deliver services to the network based on business priority and to simplify device deployment. Zero-touch device provisioning and software image management features reduce device installation or upgrade time.
- **Assurance** - Telemetry and notification for application performance and user connectivity events in real-time.
- **Platform** - Information and documentation for the DNA Center Intent API supporting the use of third-party applications and processes for data gathering and control via the API. This is the means to improve and automate IT operations, establish customized and automated workflow processes, and integrate network operations into third-party applications.

Role-based access control (RBAC)

The initial user, 'admin', is assigned the SUPER-ADMIN-ROLE allowing complete control of the DNA Center and access to all sections.

Administrators for each Cisco DNA Center may create additional 'role-based' users. The assigned role controls the level and type of access to sections of the GUI as well as API service rights.

As users are created, they are assigned a SUPER-ADMIN-ROLE, NETWORK-ADMIN-ROLE, or OBSERVER-ROLE. Users assigned a NETWORK-ADMIN-ROLE have general access and may create and manage devices. OBSERVER-ROLE assigned users have read-only access in both the GUI and API (GET only) and are restricted from some portions of the GUI interface.

Product hardware

The Cisco DNA Center appliance runs on dedicated Cisco Rack Servers in various scalable configurations. See Cisco DNA Center Data Sheet for the most recent information on hardware, environmental requirements, and scaling limits.

It is used to monitor, command and control Cisco routers, switches, wireless access points, Cisco Enterprise NFV Infrastructure Software (NFVIS) platforms, and integrated non-Cisco equipment. See Cisco DNA Center Supported Devices for compatibility information.

Cisco DNA Center Intent API

The Intent API provides the means to programmatically access the Cisco DNA Center services and functionality. This allows for automation and standardization of common management activities, as well as integration with third-party or enterprise applications. It is a RESTful API and uses HTTPS verbs (GET, POST, PUT, and DELETE) with JSON structures to discover and control the network.

Domains and subdomains

The Intent API is organized hierarchically into functional groups known as domains and subdomains.

Domain	Domain Purpose	Subdomains
Authentication	User authentication and session token generation	Authentication
Know Your Network	Discover and manage sites, topology, devices, users, and issues.	• Sites • Topology • Devices • Clients • Users • Issues
Site Management	Enterprise network provisioning, onboarding, and deployment, and software image management.	• Site Design • Network Settings • Software Image Managem • Device Onboarding (PnP) • Configuration Templates
Connectivity	Manage and configure SD-Access fabric wired and non-fabric wireless networks, including Enterprise SSIDs, wireless profiles, RF profiles, and access points.	SDA Non-Fabric Wireless
Operational Tasks	Command (CLI). Task, and Tag Management Network Discovery and Path Trace › Task Management Tag Management	• Command Runner • Network Discovery • Path Trace • File • Task • Tag
Policy	Applications, Application Sets, and Application Policy Management	Application Policy
Event Management	Configure and manage event-based notification to external handlers	Event Management

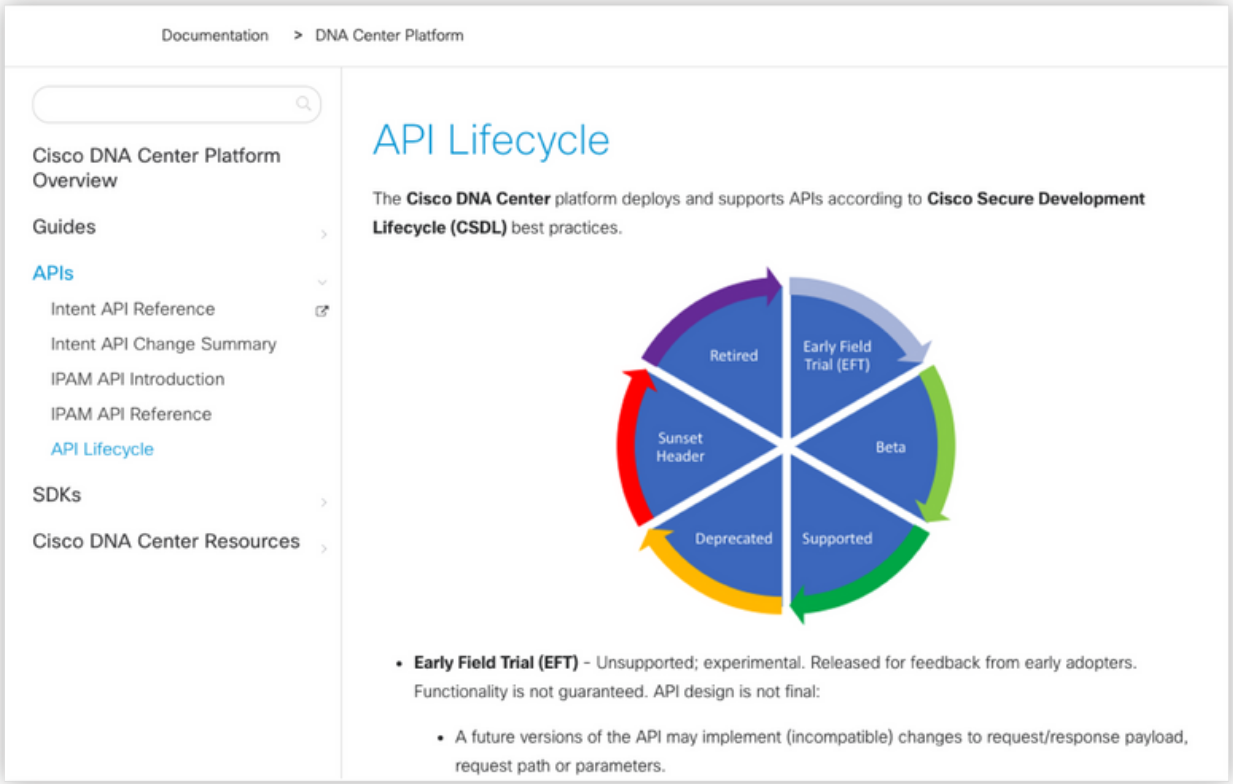
8.4.4

Cisco DNA Center – Intent API

Intent API documentation

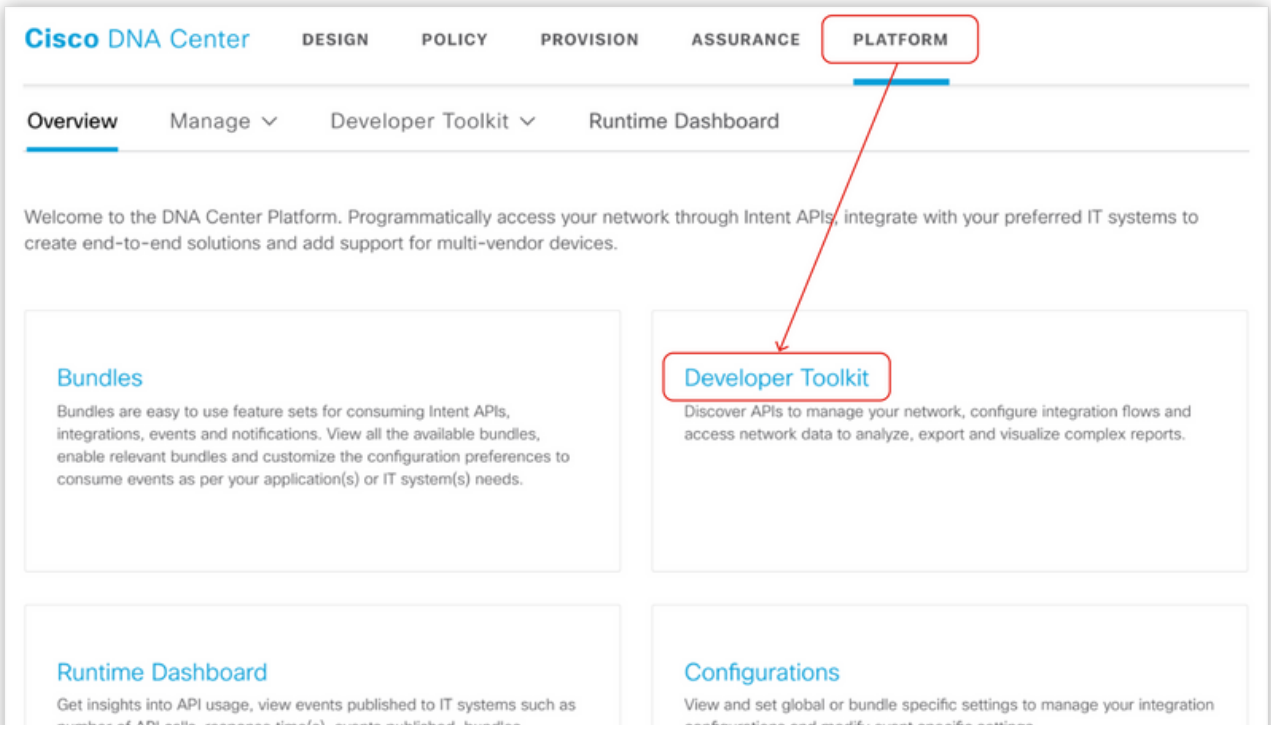
The Intent API documentation is found in the DevNet Cisco DNA Center, under section "Cisco DNA Center – Intent API".

Cisco DNA Center API documentation on DevNet



Additional documentation and a built-in 'TryIt' capability is available in the product GUI, under **Platform › Developer Toolkit** to users assigned a 'NETWORK-ADMIN-ROLE' or 'SUPER-ADMIN-ROLE'.

Cisco DNA Center API documentation via Developer Toolkit

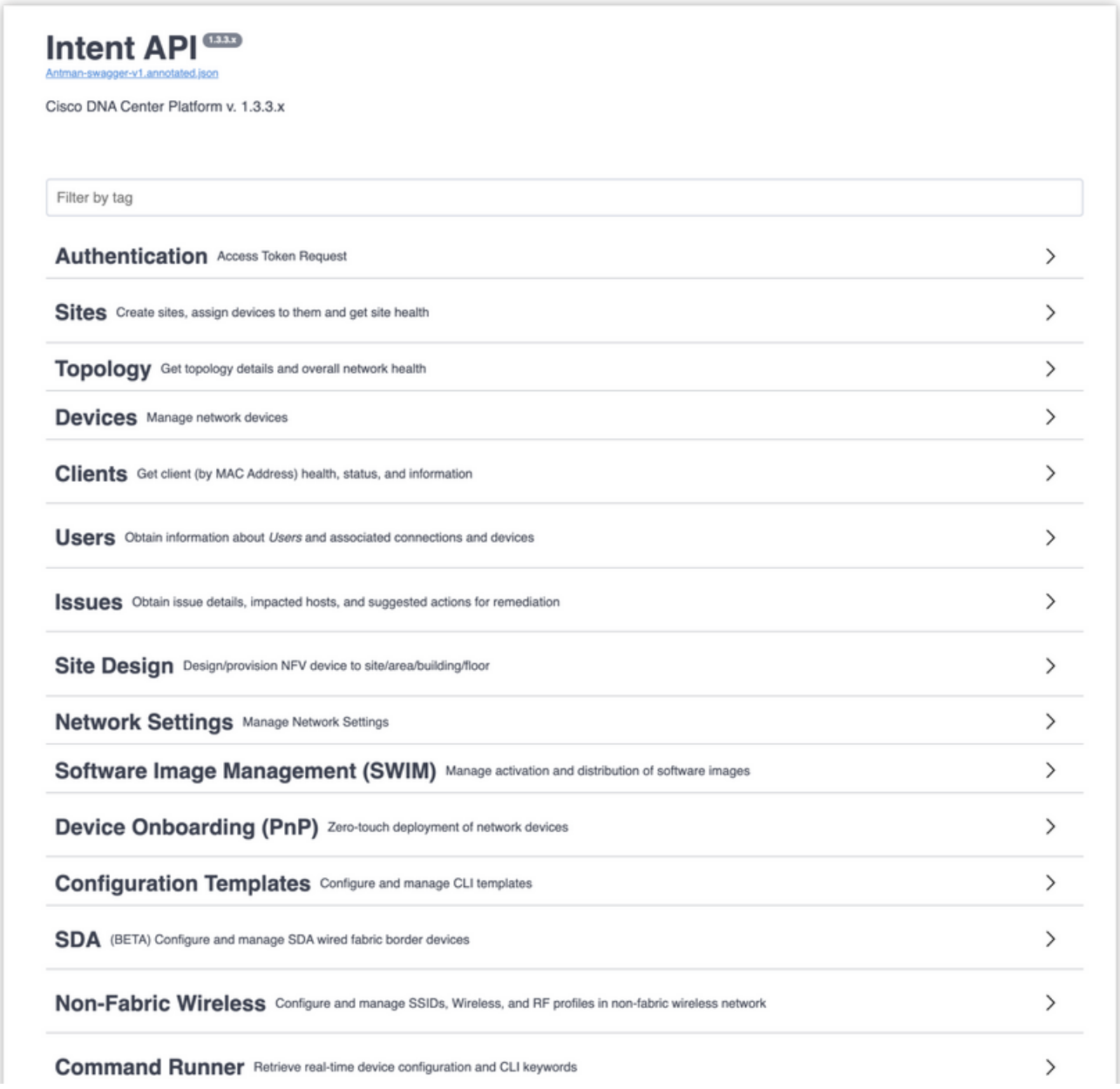


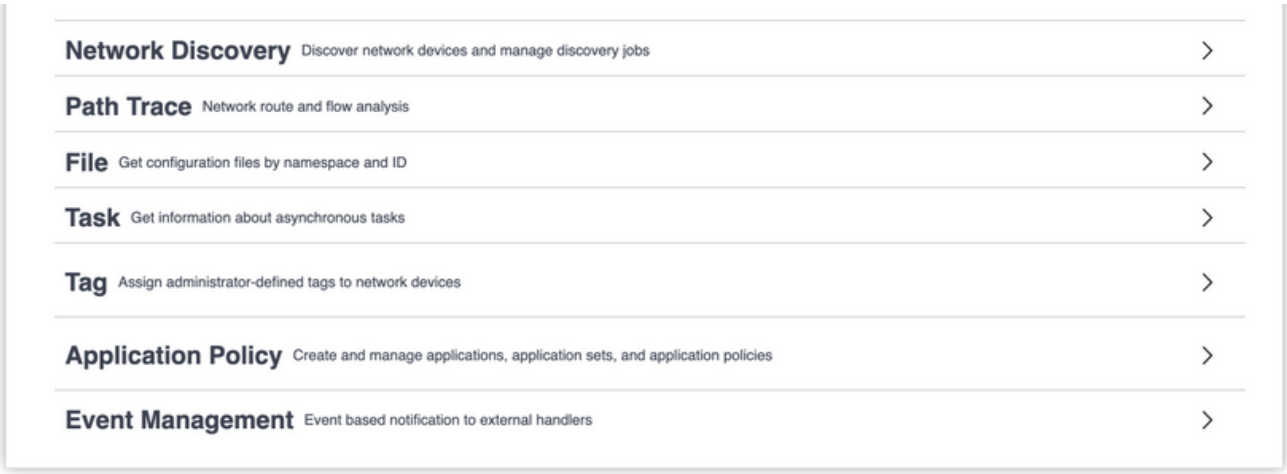
Method description, URI, and parameters

Both documentation sources list all available API methods and provide a description, request parameters, response structures, and the range of potential HTTP response codes.

(The DevNet documentation is organized by subdomain groups. The GUI Platform Developer Toolkit is organized by Domain: Subdomain.)

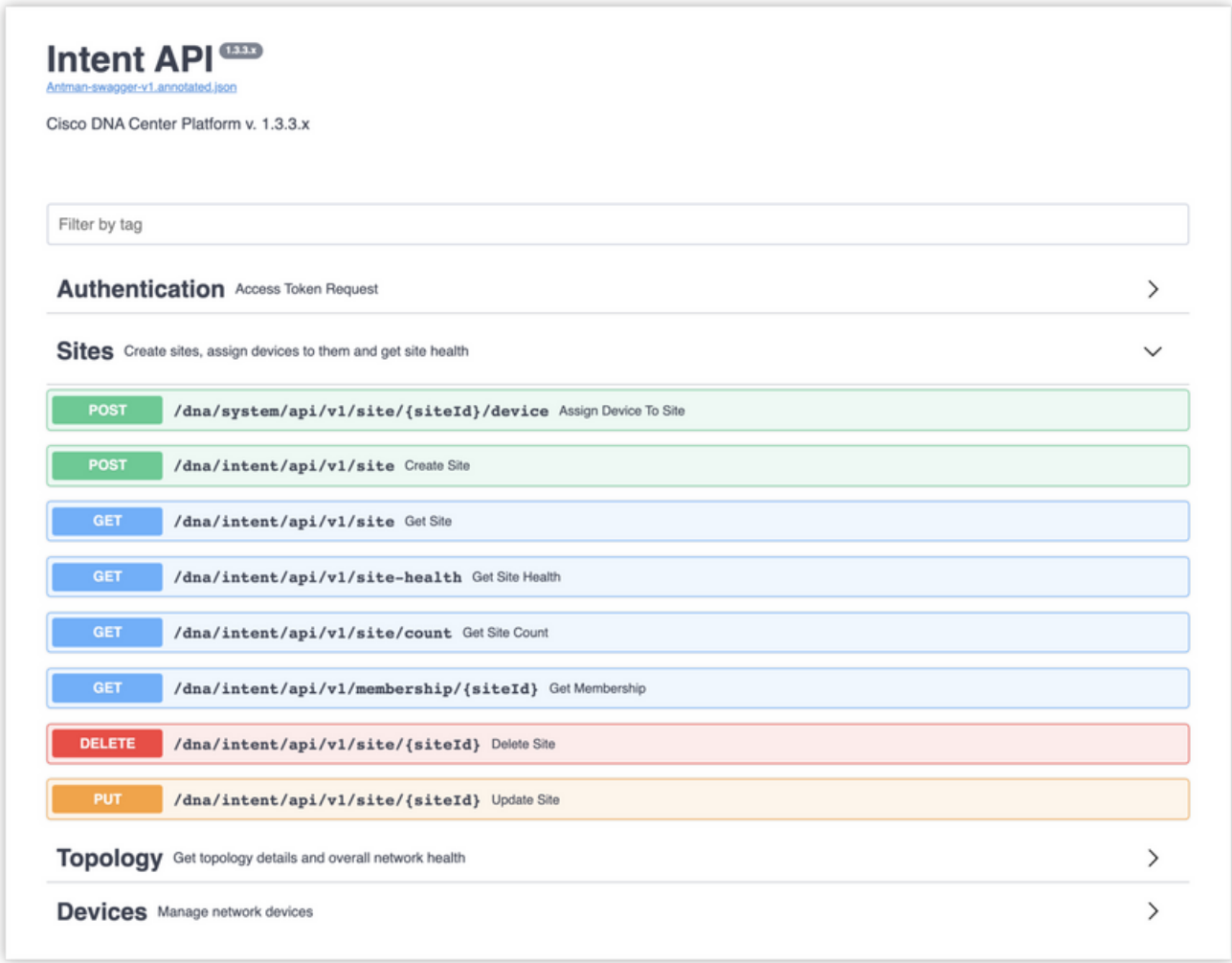
DevNet Intent API Subdomain





Clicking any of the subdomain names expands to display list of methods associated with that subdomain.

DevNet Intent API Site Methods



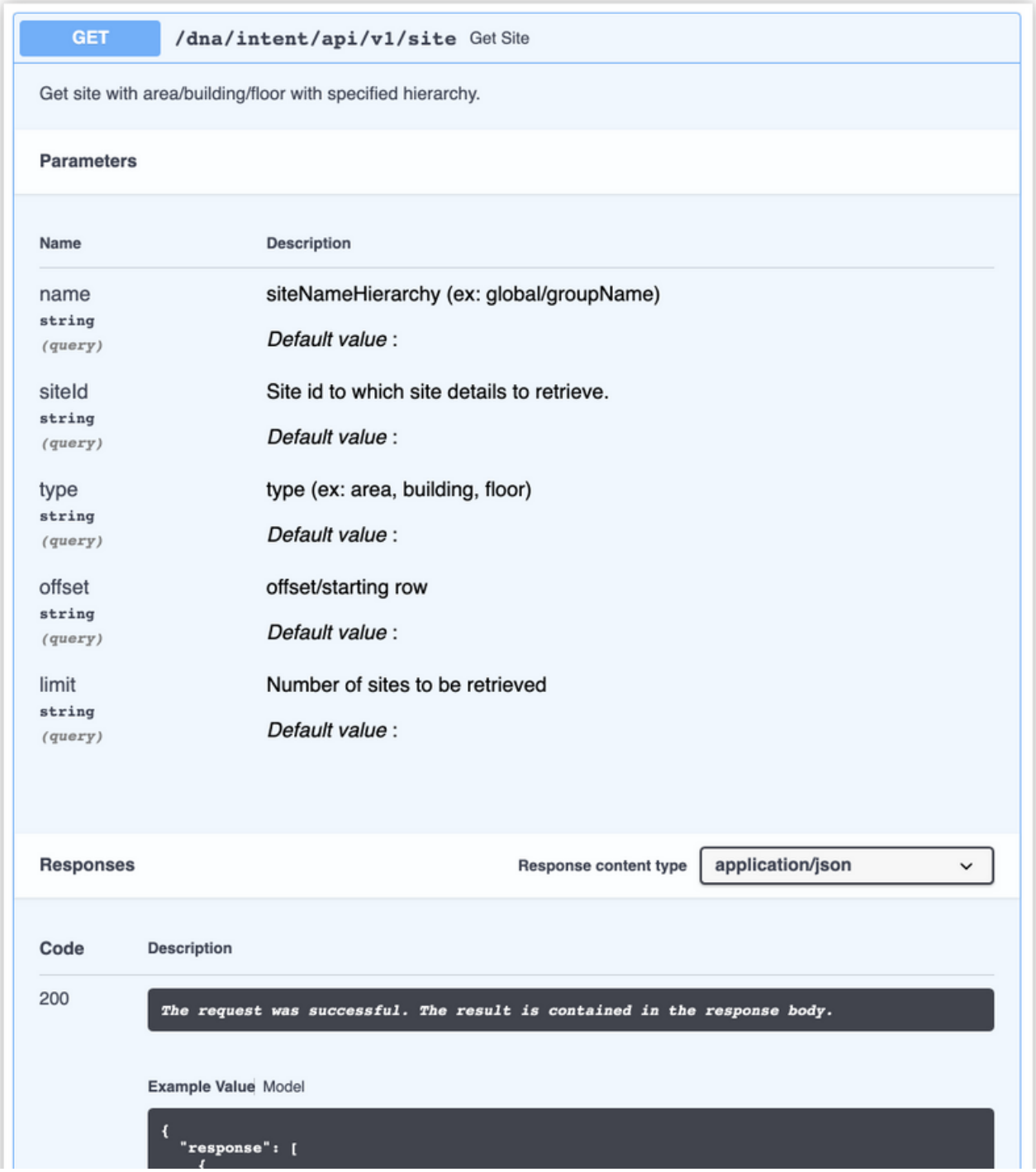
Click any of the individual methods to see documentation for that specific method. Each API will be named and described, with request parameters and structured shown and documented, and response structures

and response codes documented.

The two documentation sets display this information slightly differently. In the DevNet method documentation, the method URI is given in an abbreviated format, showing only the URL suffix. In code, when actually issuing the API call, the URI must be prefixed with the specific DNA Center URL. The GUI Platform Developer Toolkit includes the specific expanded URL using the IP address of the DNA Center.

Documentation for the **GET Site** method is shown below, as seen from both DevNet and GUI Platform: Developer Toolkit.

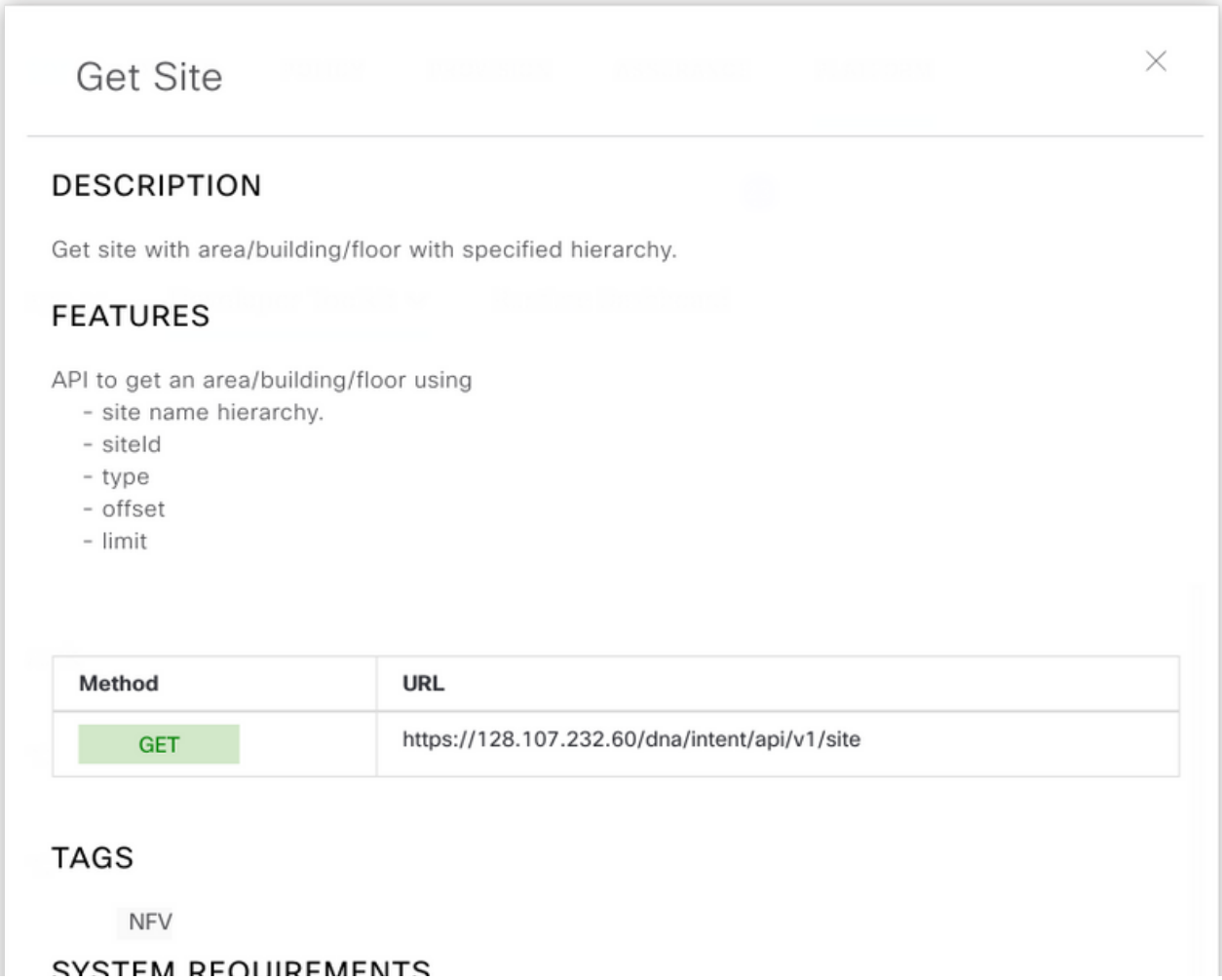
DevNet Get Site Method Detail

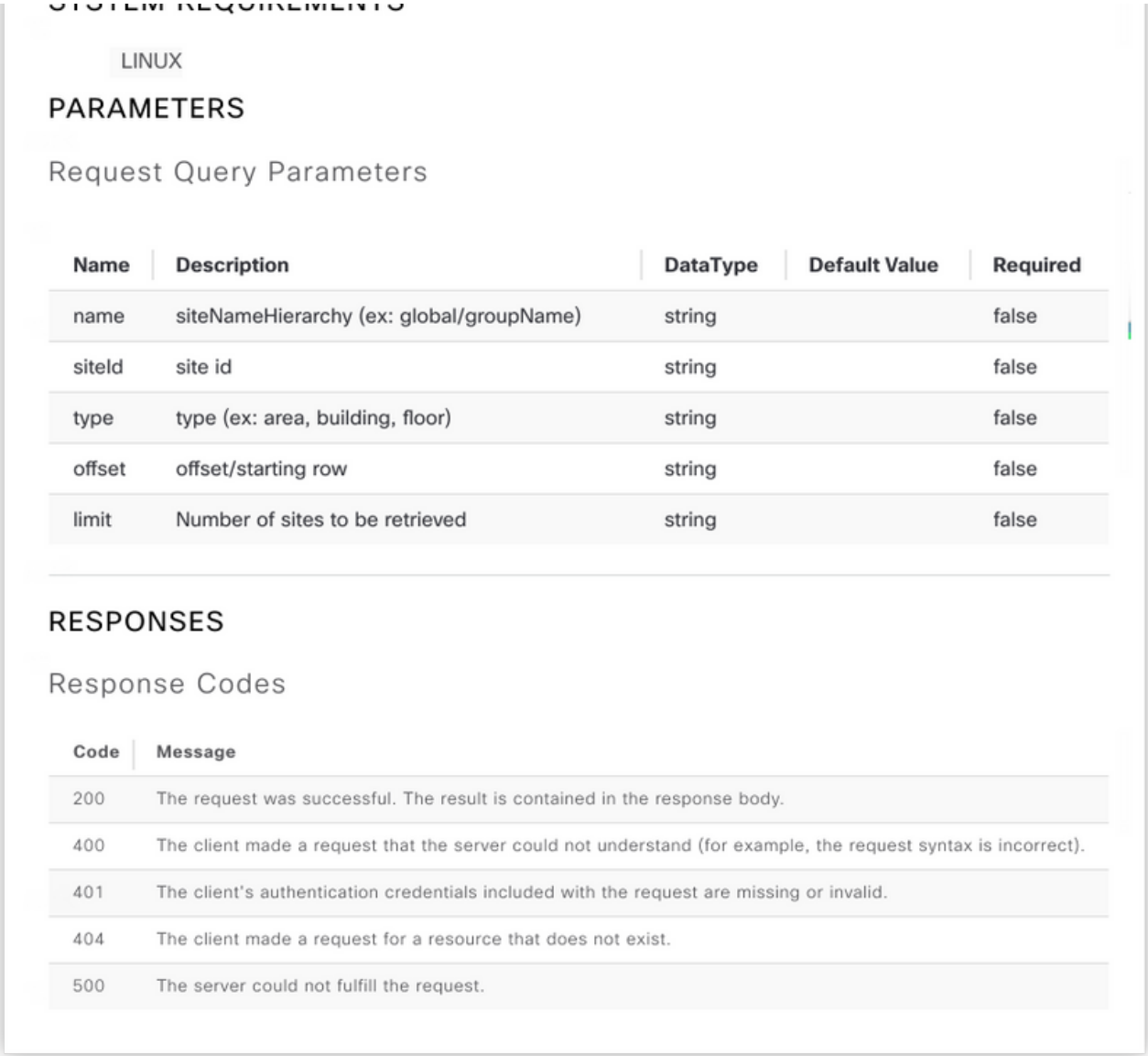


```
      "parentId": "string",
      "name": "string",
      "additionalInfo": [
        "string"
      ],
      "siteHierarchy": "string",
      "siteNameHierarchy": "string",
      "instanceTenantId": "string",
      "id": "string"
    }
  ]
}
```


400	*The client made a request that the server could not understand (for example, the request syntax is incorrect).*
401	*The client's authentication credentials included with the request are missing or invalid.*
404	*The client made a request for a resource that does not exist.*
500	*The server could not fulfill the request.*

GUI Platform Developer Toolkit Get Site Method Detail





LINUX

PARAMETERS

Request Query Parameters

Name	Description	DataType	Default Value	Required
name	siteNameHierarchy (ex: global/groupName)	string		false
siteId	site id	string		false
type	type (ex: area, building, floor)	string		false
offset	offset/starting row	string		false
limit	Number of sites to be retrieved	string		false

RESPONSES

Response Codes

Code	Message
200	The request was successful. The result is contained in the response body.
400	The client made a request that the server could not understand (for example, the request syntax is incorrect).
401	The client's authentication credentials included with the request are missing or invalid.
404	The client made a request for a resource that does not exist.
500	The server could not fulfill the request.

All Intent API methods respond with a JSON-structured content payload. Each response structure is described in the documentation.

An HTTP response code indicates success or failure for the request.

8.4.5

Cisco DNA Center – Create (POST) - Update (PUT)

Intent API PUT and POST methods require a JSON request payload. These are documented in each method description.

Both POST and PUT requests are handled within the Cisco DNA Center asynchronously, which means that the request to add, create, or modify a resource is initiated with a correctly formed request, but not necessarily completed before responding. A request which has been successfully initiated, responds with a structure containing `executionId`, `executionStatusUrl`, and a status `message`.

Completion status is obtained using Task API methods to monitor execution completion. File API methods are used to obtain additional results following successful execution completion.

This part of the module works with 'read-only' GET methods only. POST, PUT, and DELETE methods that modify the target network are beyond the scope of this part of the course.

Documentation

Documentation for the Cisco DNA Center is organized into multiple separate guides. For more information consult the relevant published guides.

See Cisco DNA Center User Guides for the latest revisions of the following:

- Cisco DNA Center User Guide
- Cisco DNA Assurance User Guide
- Cisco DNA ITSM Integration Guide
- Cisco DNA Center Platform User Guide

See Cisco DNA Center Maintain and Operate Guides for the latest revision of the following:

- Cisco Digital Network Architecture Center Administrator Guide
- Cisco DNA Center High Availability Guide

See Cisco DNA Center Install and Upgrade Guides for the latest revision of the following:

- Cisco DNA Center Second Generation Appliance Installation Guide
- Cisco DNA Center First Generation Appliance Installation Guide
- Cisco DNA Center Upgrade Guide

8.4.6

Video – Access Cisco DNA Center Using REST APIs in Python

8.4.7

Cisco ACI

The Cisco Application Centric Infrastructure (ACI) platform, run on Nexus 9000 hardware, is the Cisco solution for Software-Defined Networking (SDN). The centralized management system is the Application Policy Infrastructure Controller (APIC), a cluster of controllers. The APIC manages and provides for automation and management, policy programming, application deployment, and health monitoring for the fabric. With the APIC, you get unified operation of both physical and virtual, or software-based infrastructure. This platform helps network engineers and developers to manage networks programmatically.

Instead of opening up a subset of the network functionality through programmatic interfaces, the entire ACI infrastructure is opened up for programmatic access. This entirety is achieved by providing access to Cisco ACI object model. The ACI object model represents the complete configuration and run-time state of every software and hardware component in the entire infrastructure. The object model is made available through standard REST API interfaces, making it easy to access and manipulate the configuration and run-time state of the system. The API accepts and returns HTTP or HTTPS messages that contain JSON or XML documents. You can use any programming language to generate messages and JSON or XML documents that contain the API methods or Managed Object (MO) descriptions.

A common scenario for network automation is to manage the network with the same tools as you do other infrastructure. Combining the ACI data models with DevOps tools like Ansible, Chef, or Puppet, you can manage your infrastructure holistically with applications in mind. You can also build integrations with other management, security, policy, and monitoring tools.

Note: Rather than ACI mode, Nexus 9000 switches may be configured in NX-OS mode to access device-level APIs. In NX-OS mode you can manage switches as a Linux device.

ACI use cases

Common use cases include:

- Programmability as a single fabric, with access to read and write object models representing all attributes in the system.
- Desired state defined and enforced.
- Extension to AWS or Azure public clouds (via Multi-Site Orchestrator (MSO) and its API)

An example walkthrough could include providing a check of a particular tenant, continuously monitoring whether it changes, and if it changes, update the configuration back to the desired state.

Tools used with ACI

In addition to the standard REST interface, Cisco provides several open source tools or frameworks such as the ACI toolkit, Cobra (Python), ACIrb (Ruby), Puppet, and Ansible to automate and program the APIC. On top of REST API are also both a CLI and a GUI for day-to-day administration.

Note: The above tools are available for accessing the Multi-Site Orchestrator (MSO) for hybrid clouds as well. MSO's API is different from ACI's, the latter is used to access a private ACI fabric via its Application Policy Interface Controller (APIC).

Cobra (Cisco ACI) Python SDK

The Cobra SDK gives you access to all REST functions using native Python bindings. Objects in Cobra are one-to-one representations of the Management Information Tree (MIT). Cobra installation documentation can be found at http://cobra.readthedocs.org. The SDK offers Python methods to match the REST methods that the GUI uses, such as those shown in the API Inspector. It offers full functionality and is suited for complex queries, incorporating Layer 4-7 devices, initial fabric builds, and so on.

Cisco ACI toolkit

This toolkit is a set of Python libraries you can use for basic configuration of a subset of the object model. Available on GitHub at https://github.com/datacenter/acitoolkit. This set exposes a small subset of the Cisco APIC object model, unlike the full functions of the Cobra SDK.

APIC REST Python adapter (ARYA)

This tool converts XML/JSON objects to equivalent Python code. Often people use it with the API Inspector built into the product. Note that the tool does not validate targets or perform lookups. Documentation is available at https://developer.cisco.com/docs/aci/#!arya.

ACIrb

This tool provides a Ruby-based implementation of the Cisco APIC REST API. It enables direct manipulation of the MIT through the REST API using standard Ruby language patterns.

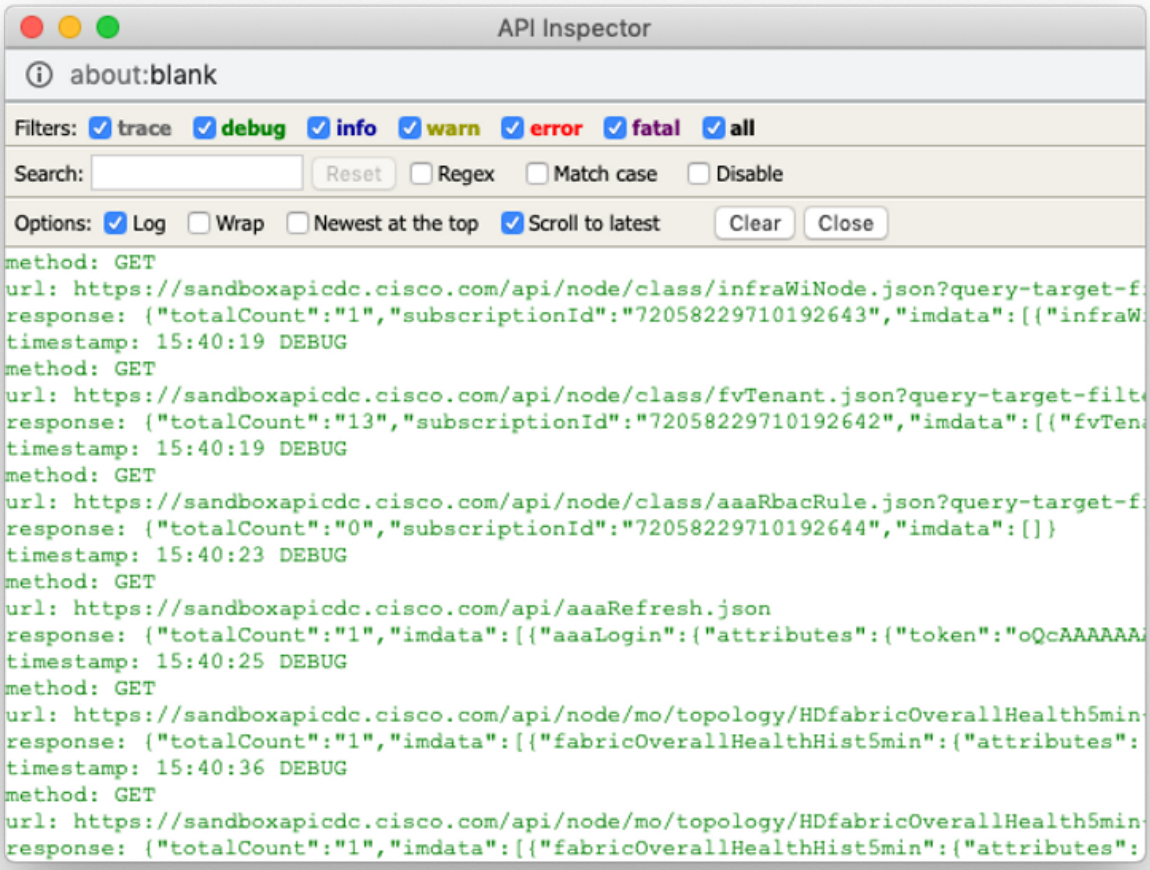
CLI

ACI also offers an NX-OS style CLI to configure and manage ACI in a traditional CLI way. Moquery is a CLI Object Model query tool, while Visore is Object Store Browser (GUI).

API Inspector

When you perform a task in the Cisco APIC GUI, the GUI creates and sends internal API messages to the operating system to execute the task. By using the API Inspector, which is a built-in tool of the Cisco APIC, you can view and copy these API messages. An administrator can replicate these messages to automate key operations, or you can use the messages as examples to develop external applications that will use the API.

APIC API Inspector



8.4.8

Cisco ACI – Ansible Modules

Ansible modules for ACI (and MSO)

Cisco and the wider community have collaborated on a broad suite of open source modules for Ansible, enabling configuration and management of ACI fabrics as code, alongside other Ansible-managed inventory. Modules have also been created to address multi-site and hybrid cloud resources via the Multi-Site Orchestrator (MSO) APIs. Ansible provides a complete index of modules with links to documentation.

The ACI/MSO modules permit the simple creation of playbook elements to perform inquiry, administration, and management tasks upon an ACI fabric. For example, the following task, drawn from Ansible documentation, idempotently ensures that a given tenant account exists (creating it, if not). It uses the aci_tenant module, which provides a range of tenant-management functionality.

```
- name: Ensure tenant customer-xyz exists
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password
    tenant: customer-xyz
    description: Customer XYZ
    state: present
```

Cisco ACI REST API Example

With the REST API you have an interface into the MIT, allowing you to manipulate the object model state. The APIC CLI, GUI, and SDK use the same REST API interface so that whenever information is displayed, the data is read through the REST API, and when configuration changes are made, they are written through the REST API. Additional data available with the API includes statistics, faults, and audit events. With WebSockets, the API even provides a means of subscribing to push-based event notification. When a change occurs in the MIT, the API can send an event notification.

The APIC REST API supports standard methods such as POST, GET, and DELETE operations through HTTP. The POST and DELETE methods are idempotent, meaning an operation can be repeated without additional effects when called with the same input parameters. The GET method is nullipotent, meaning that performing it multiple times has the same effect as performing it zero (null or never) times. Basically, the GET call is a read-only operation.

The REST interface accepts payloads to and from the resources with either XML or JSON encoding. For XML, the encoding operation is simple: the element tag is the name of the package and class, and any properties of that object are specified as attributes of that element. You create child elements in XML to define nested containment in a model.

Any data on the MIT can be described as a self-contained structured text tree document, encoded in XML or JSON. With that information model, Cisco ACI fits in well for REST API design: URLs and URIs map directly to Distinguished Names (DNs) that identify objects on the tree, and the objects have parent-child relationships also identified using distinguished names and properties. This makes the REST API useful to read and modify data with a set of Create, Read, Update, Delete (CRUD) operations.

You access these objects for retrieval and manipulation of Cisco APIC object data, using standard HTTP commands and a well-defined REST URL.

Distinguished name (DN) - Identifies a specific target object, letting you map the name directly to URLs.

Relative name (RN) - Names the object apart from its siblings within the context of a parent object.

Object instances are referred to as Managed Objects (MOs). Every MO in the system can be identified by a unique DN. With the MO and its DN, you can refer to any object globally. In addition to a DN, you can refer to each object by its RN. The RN identifies an object relative to its parent object.

You can use either the RN or the DN to access an object, depending on the current location in the MIT. You can query the tree in several ways because it is hierarchical. Then combine hierarchy with the attribute system for further object identification. You can perform a query for an object through its DN, or on a class of objects such as a switch chassis, or on a tree-level to discover all related members of an object.

8.4.9

Cisco ACI URL Format

ACI URL Format

It's probably easier to understand with a few worked examples. Envision the URL format representation as follows:

- **http:// | https://**: By default, only HTTPS is enabled.
- **host**: This component is the hostname or IP address of the APIC controller, for example, "SandboxAPIC".
- **port**: This component is the port number for communicating with the APIC controller if a nonstandard port configured.
- **api**: This literal string (`api`) specifies that the message is directed to the API.
- **mo | class**: This component specifies the target of the operation as a managed object or an object class.

- **DN**: This component is the DN of the targeted managed object, for example, topology/pod-1/node-201.
- **className**: This component is the name of the targeted class, concatenated from the package and the class in the context of the package, for example, `dhcp:Client` is `dhcpClient`. The className can be defined in the content of a DN, for example, `topology/pod-1/node-1`.
- **json | xml**: This component specifies the encoding format of the command or response body as JSON or XML.
- **?options**: This component includes optional filters, selectors, or modifiers to a query. Multiple option statements are joined by an ampersand sign (`&`).

A URI provides access to a target resource. The first two sections of the request URI specify the protocol and access details of the APIC. The literal string `api`, indicates that the API is to be invoked. The next section of the URI path specifies whether the operation is for a Managed Object or a class.

Next in the path is either the fully qualified DN for object-based queries, or the package and class name for class-based queries. The final mandatory part of the request URI is the encoding format: either `.xml` or `.json`.

The REST API supports a wide range of flexible filters, useful for narrowing the scope of your search to allow information to be located more quickly. The filters themselves are appended as query URI options, starting with a question mark (`?`) and concatenated with an ampersand (`&`). Join multiple conditions when you want to form complex filters.

With the capability to address an individual object or a class of objects with the REST URL, the system provides complete programmatic access to the entire object tree and to the entire system.

One of the most common use cases for this API is for monitoring the Cisco ACI Fabric. Proactive monitoring is a very important piece of the network administrator's job, but is often neglected because putting out fires in the network usually takes priority. However, because the APIC makes it incredibly easy to gather statistics and perform analyses, it saves network administrators both time and frustration.

For example, if you want to learn about the details of all available fabric nodes (Cisco ACI leaf and spine switches) at host `devasc-aci-1.cisco.com`, including the state, IP address and so on, you could use the following Cisco ACI REST API call:

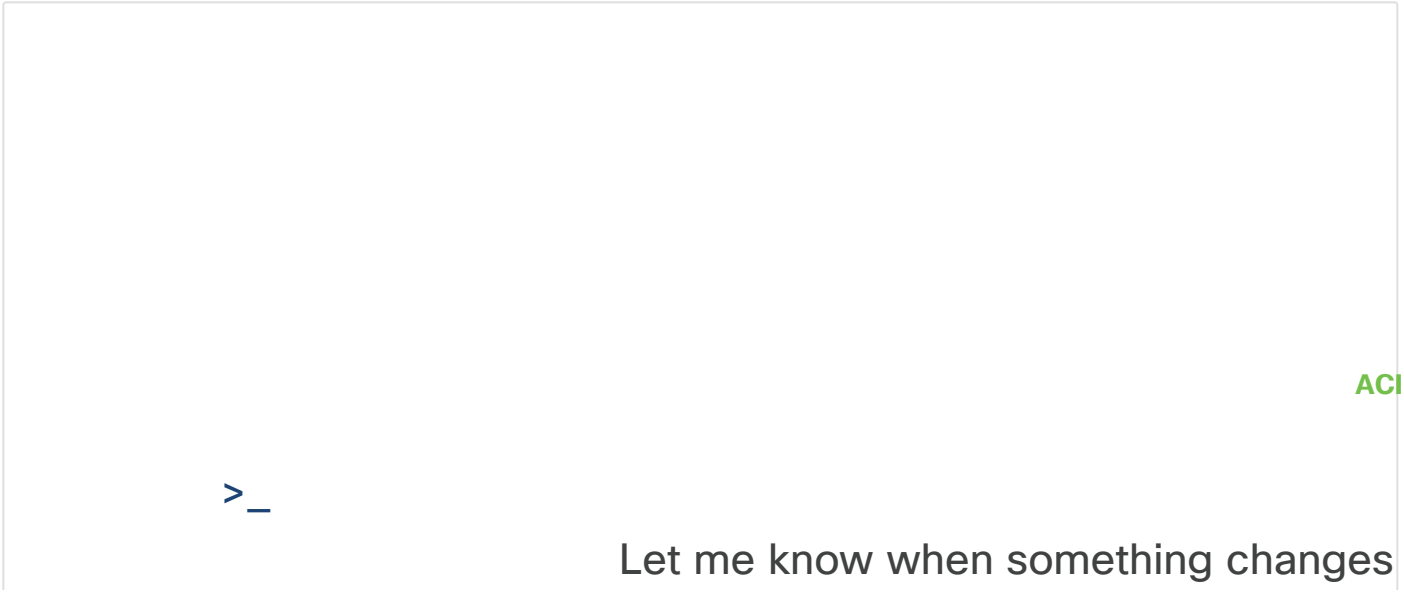
GET https://devasc-aci-1.cisco.com/api/node/class/fabricNode.json

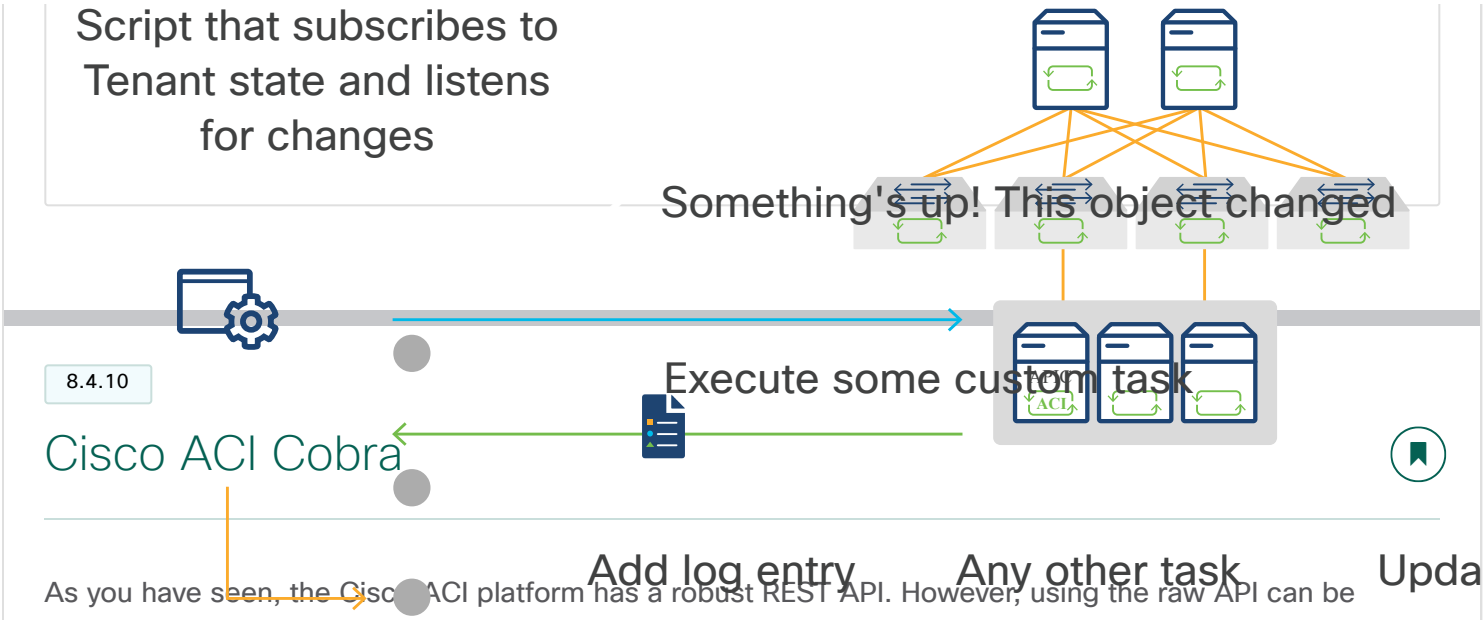
```
Response code: 200
Response body:
{
    "totalCount": "6,"
    "imdata": [
        {
            "fabricNode": {
                "attributes": {
                    "adSt": "on,"
                    "address": "10.0.240.32,"
```

```
                    "annotation": ","
                    "apicType": "apic,"
                    "childAction": ","
                    "delayedHeartbeat": "no,"
                    "dn": "topology/pod-1/node-101,"
                    "extMngdBy": ","
                    "fabricSt": "active,"
                    "id": "101,"
                    "lastStateModTs": "2019-11-17T15:32:30.294+00:00,"
                    "lcOwn": "local,"
                    "modTs": "2019-11-17T15:32:53.511+00:00,"
                    "model": "N9K-C9396PX,"
                    "monPolDn": "uni/fabric/monfab-default,"
                    "name": "leaf-1,"
                    "nameAlias": ","
                    "nodeType": "unspecified,"
                    "role": "leaf,"
                    "serial": "TEP-1-101,"
                    "status": ","
                    "uid": "0,"
                    "vendor": "Cisco Systems, Inc,"
                    "version": ""
                }
            }
        },
        ...
```


Besides manual querying, you can gather information automatically and then apply policies, minimizing the opportunities for human error. As you can see in the figure below, when you start using automation when monitoring the ACI fabric, you can build various applications that can execute different tasks if there is a specific change in the network.

Proactive Monitoring of the ACI Fabric





As you have seen, the Cisco ACI platform has a robust REST API. However, using the raw API can be tedious and cumbersome. This is because you need to know and configure low-level details such as which HTTP verb is being used, the URL, headers, and encoding supported. Additionally, it would be important to code error handling in custom code using a native REST API.

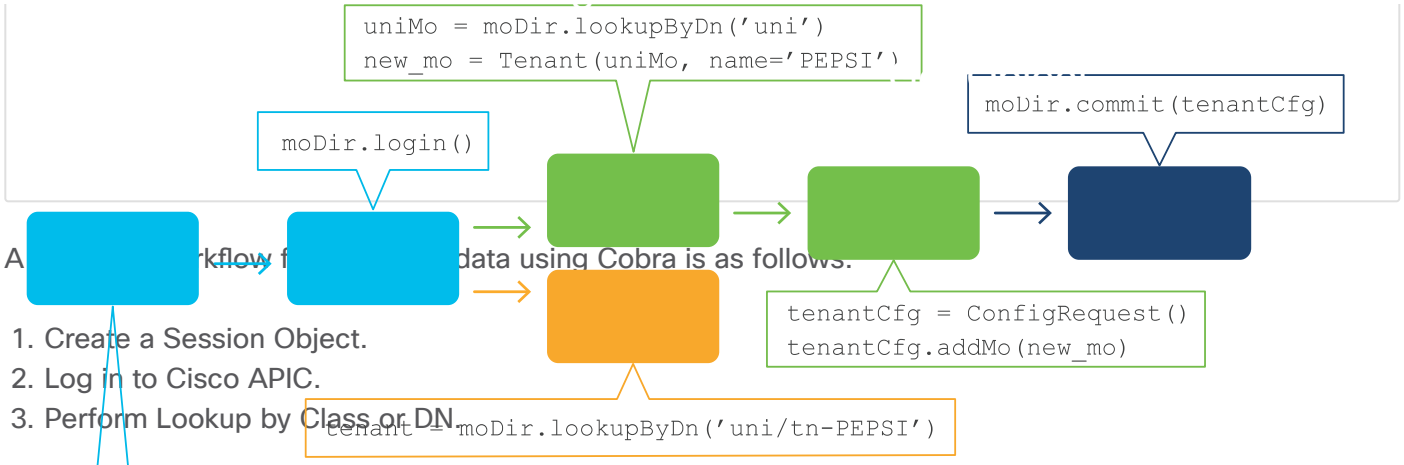
To simplify application development with ACI, Cisco has developed Cobra, a robust Python library for the APIC REST API. Objects in the Cobra library (SDK) map 1:1 to the objects within the Cisco ACI MIT. If you are planning to dive deeper into Cobra, the best place to start is the official Cobra documentation. These documents review everything from installation to showing examples, and even include a Frequently Asked Questions section to jump-start individuals looking to test Cobra.

To access the APIC using Cobra, you must log in with valid user credentials. Cobra currently supports username/password-based authentication, in addition to certificate-based authentication. To make configuration changes, you must have administrator privileges in the domain in which you will be working. A successful login returns a reference to a directory object that you will use for further operations.

You can use the Cobra SDK to manipulate the MIT generally through this workflow:

- Identify the object to be manipulated.
- Build a request to read, change attributes, or add or remove children.
- Commit the changes made to the object.

Workflow of the Cobra SDK Objects



A [illegible]rkflow f[illegible]data using Cobra is as follows:

1. Create a Session Object.
2. Log in to Cisco APIC.
3. Perform Lookup by Class or DN.

A common workflow for configuring the Cisco ACI Fabric is as follows:

1. Create a Session Object.
2. Log in to Cisco APIC.
3. Create a Configuration Object.
4. Create a Configuration Request Object.
5. Add your Config Object to the Request.
6. Commit.

Summary

You now have some powerful tools to investigate and use with Cisco ACI. With an underlying REST API and standard data model plus hierarchy, you can manage and monitor a myriad of infrastructure, both virtual and hardware-based.

8.4.11

Video - Access a Cisco APIC Using REST APIs in Python

8.4.12

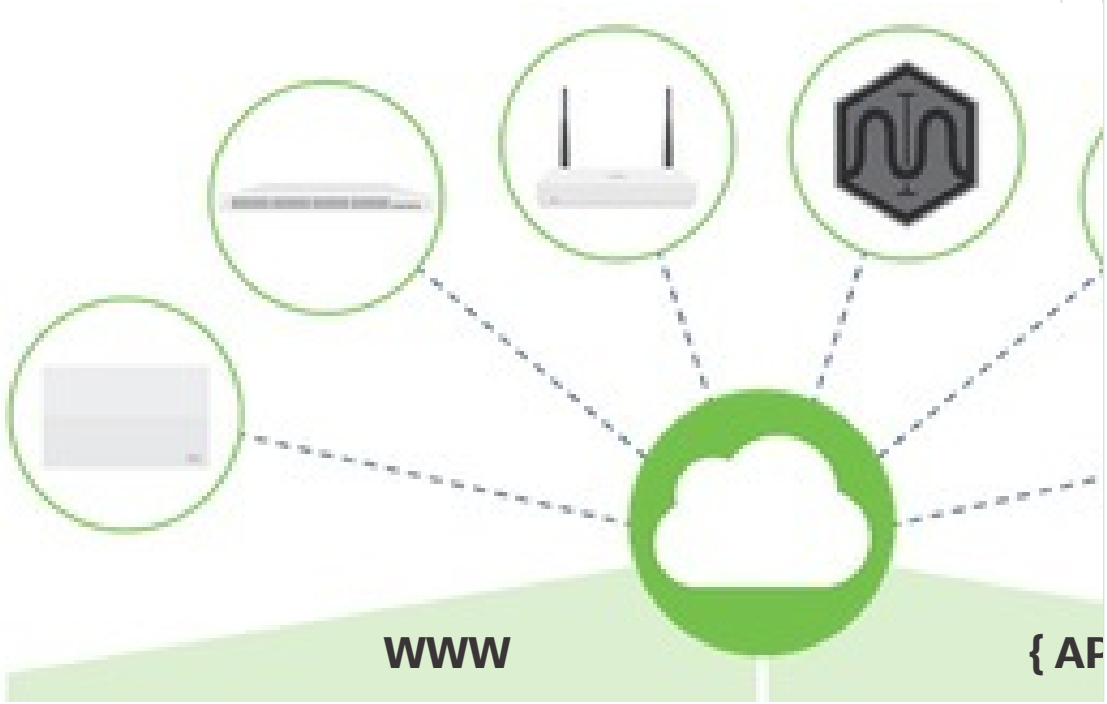
Cisco Meraki

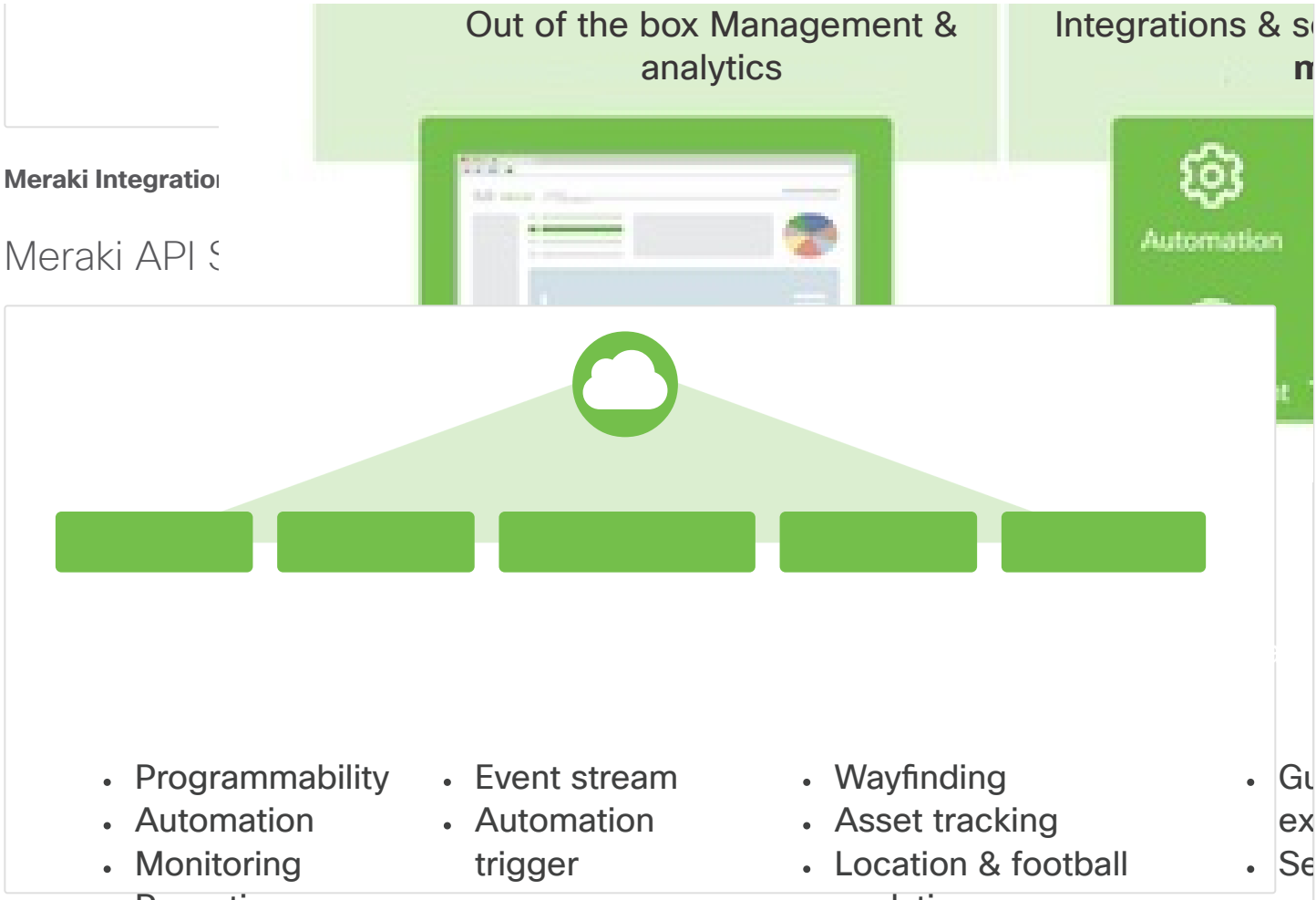
Cisco Meraki is a suite of cloud-managed network solutions that enables a single source of management for infrastructure, locations, and devices.

Components include:

- Integration APIs
- A complete cloud-managed network infrastructure solution
- Wireless, switching, security, SD-WAN, Mobile Device Management (MDM), and smart cameras
- Integrated hardware, software, and cloud services

Meraki Cloud Platform





Meraki Integratio

Meraki API S

The Meraki enterprise cloud-managed networking infrastructure service has five different APIs for integration:

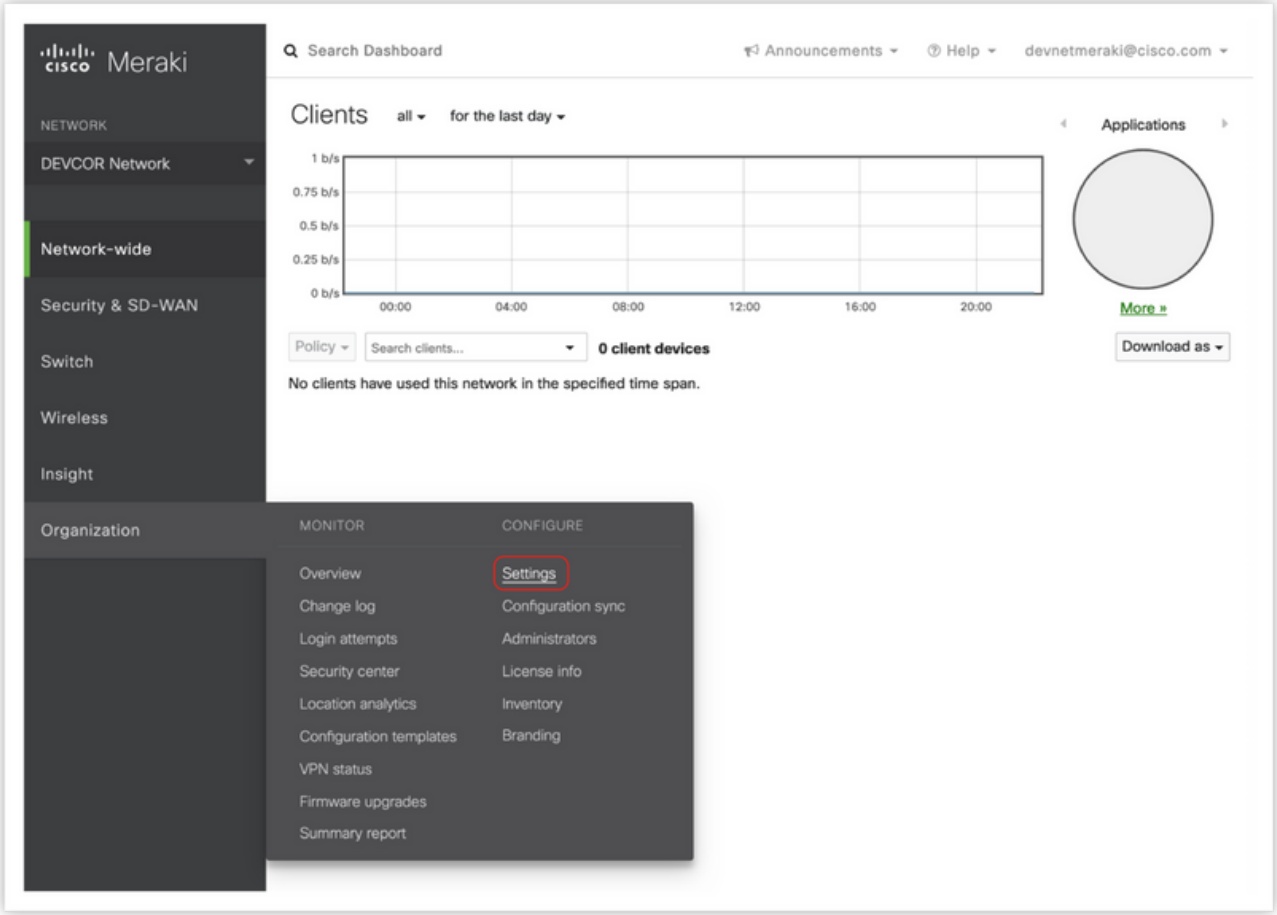
- **Dashboard API –** This a RESTful service for device provisioning, management, and monitoring.
- **Webhook API –** This is a real-time notification system for network alerts, covering events and network health.
- **Location Streaming API –** This is an **HTTP POST** method providing Wi-Fi and Bluetooth client location information (GPS, X/Y) based on Meraki Access Point (AP) map placement, and client signal strength.
- **External Captive Portal (EXCAP) API –** This enables an organization to build out custom engagement models at Wi-Fi access points.
- **MV Sense API –** This is a combination of REST APIs and realtime MQTT stream supporting oversight of a physical space.

Meraki Dashboard API

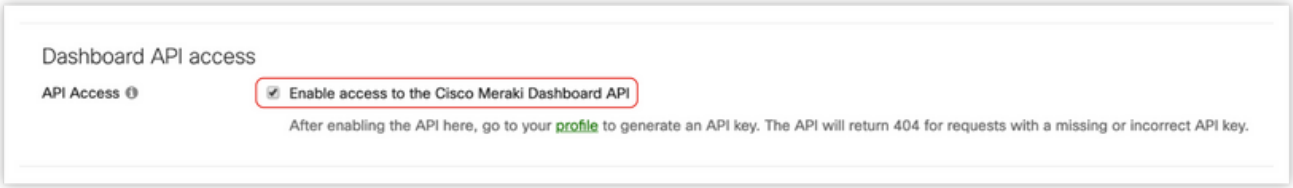
The Cisco Meraki Dashboard API is a RESTful API that uses HTTPS for transport and JSON for object serialization.

Note: These instructions are an example only. To complete these steps, you need to be a user with administrator access in a Meraki organization. The Meraki organization in this course has API access already and the API key is provided in the examples.

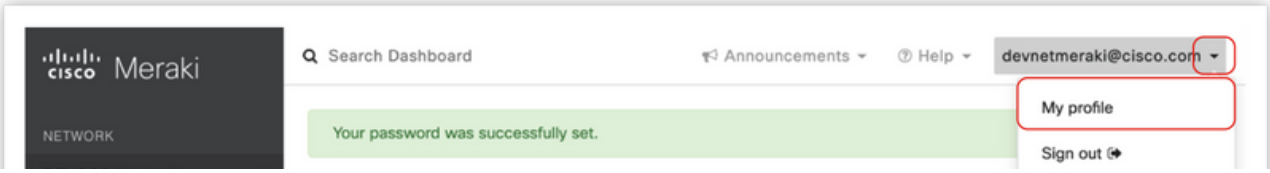
1. To provide access to the API for an organization, first enable the API for the organization under **Organization › Settings**.

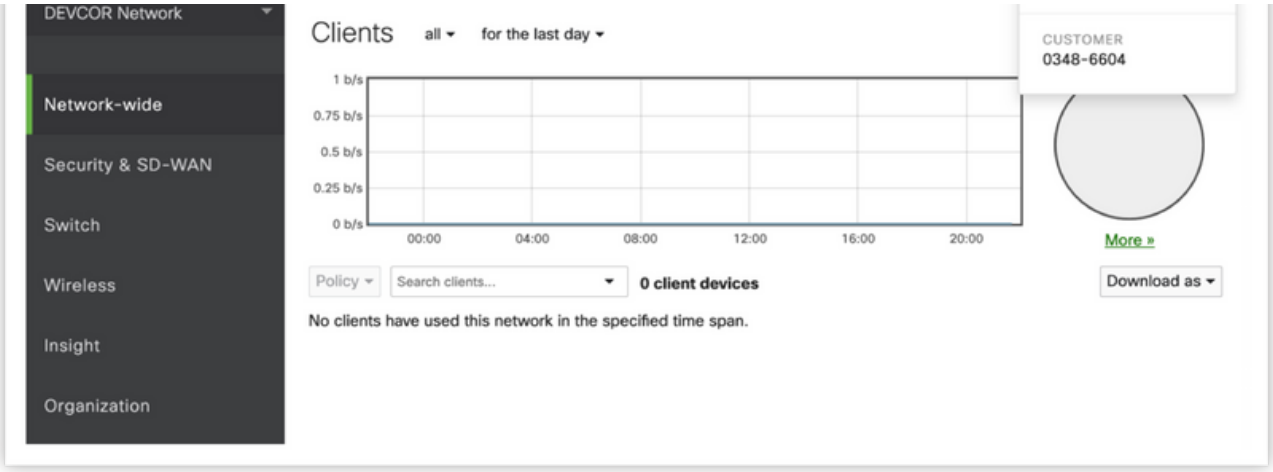


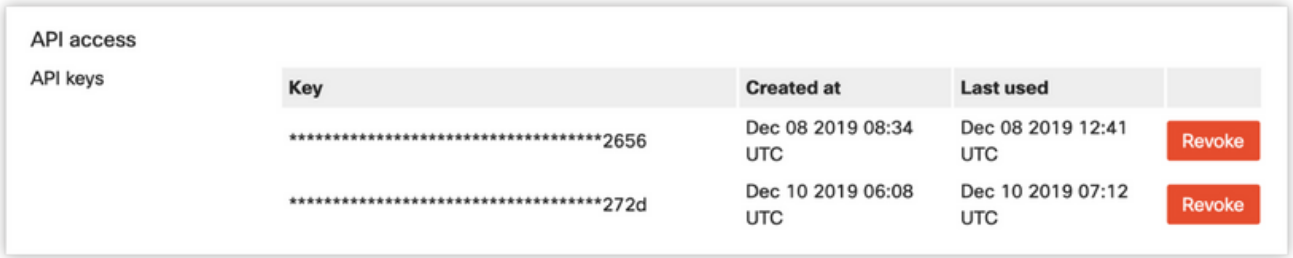
1. Scroll down.



1. After enabling the API, go to the **My Profile** page to generate an API key. The API key is associated with a Meraki Dashboard administrator account. An API key can be generated, revoked, and regenerated for the user profile. **The API key needs to be kept safe**, as it provides authentication to all of the organizations that have the API enabled. If the API key is shared, it can be regenerated at any time. Regenerating the key revokes the prior API key.







Authorization

Every request must specify an API key via a request header. The API will return a 404 (rather than a 403) in response to a request with a missing or incorrect API key. This behavior prevents leaking even the existence of resources to unauthorized users.

```
X-Cisco-Meraki-API-Key: <secret key>
```

Meraki API call examples

API call to retrieve organization list

Below is a `curl` example that retrieves the list of organizations for the API key specified in `X-Cisco-Meraki-API-Key`. You can copy and run these in your terminal if you have `curl` installed.

```
curl --request GET -L \
  --url https://api.meraki.com/api/v0/organizations \
  --header 'X-Cisco-Meraki-API-Key: 8f90ecec4fca692f606092279f203c6020ca011c'
```

The result is similar to this, though more organizations may be listed in the JSON:

```
```

```
[
   {
      "id":"566327653141842188",
      "name":"DevNetAssoc",
      "url":"https://n6.meraki.com/o/dcGsWag/manage/organization/overview"
   }
]
```

Note that the request is redirected by the `-L` argument. An HTTP 302 (Found/Redirect) response is valid for any request, including those that may modify state such as `DELETE`, `POST`, and `PUT`. Client applications must be capable of following a redirect. The `--url` parameter contains the Meraki API that you are calling. The `--header` parameter holds the API Key that authorizes the API call. These examples follow the same pattern.

API call to retrieve organization metadata

The next call is a `curl` example to retrieve metadata about a specific organization, using the ID from the result above:

```
curl --request GET -L \
  --url https://api.meraki.com/api/v0/organizations/566327653141842188 \
  --header 'X-Cisco-Meraki-API-Key: 8f90ecec4fca692f606092279f203c6020ca011c'
```

The result is:

```
{
  "id": "566327653141842188",
  "name": "DevNetAssoc",
  "url": "https://n6.meraki.com/o/dcGsWag/manage/organization/overview"
}
```

API call to list networks

The following `curl` example shows how to list the networks in an organization.

```
curl --request GET -L \
  --url https://api.meraki.com/api/v0/organizations/566327653141842188/networks \
  --header 'X-Cisco-Meraki-API-Key: 8f90ecec4fca692f606092279f203c6020ca011c'
```

The result is similar is this JSON:

```
[
    {
        "id": "L_566327653141858435",
        "organizationId": "566327653141842188",
```

```
        "name": "DevNetAssoc1",
        "timeZone": "America/Los_Angeles",
        "tags": null,
        "productTypes": [
            "appliance",
            "switch",
            "wireless"
        ],
        "type": "combined",
        "disableMyMerakiCom": false,
        "disableRemoteStatusPage": true
    },
    {
        "id": "L_566327653141858436",
        "organizationId": "566327653141842188",
        "name": "DevNetAssoc2",
        "timeZone": "America/Los_Angeles",
        "tags": null,
        "productTypes": [
            "appliance",
            "switch",
            "wireless"
        ],
        "type": "combined",
        "disableMyMerakiCom": false,
        "disableRemoteStatusPage": true
    },
    {
        "id": "L_566327653141858437",
        "organizationId": "566327653141842188",
        "name": "DevNetAssoc3",
        "timeZone": "America/Los_Angeles",
        "tags": null,
        "productTypes": [
            "appliance",
            "switch",
            "wireless"
        ],
        "type": "combined",
        "disableMyMerakiCom": false,
        "disableRemoteStatusPage": true
    },
    {
        "id": "L_566327653141858438",
        "organizationId": "566327653141842188",
        "name": "DevNetAssoc4",
        "timeZone": "America/Los_Angeles",
        "tags": null,
        "productTypes": [
```

```
            "appliance",
            "switch",
            "wireless"
        ],
        "type": "combined",
        "disableMyMerakiCom": false,
        "disableRemoteStatusPage": true
    }
]
```

Meraki webhook alerts

Webhooks provide a POST to a particular URL when a defined criterion is met. With Meraki webhooks you can set up systems that subscribe to Meraki Cloud alerts. You configure which network alerts and events can automatically trigger the next action.

Alerts sent with Meraki webhooks vary in frequency depending on different factors. There may be computation time needed, or a delay required in order to aggregate multiple events before delivering the alert. You can also configure the timing threshold value, such as waiting for five minutes of downtime before sending a message.

You can calculate the delivery time by comparing the `occurredAt` and `sentAt` parameter values.

Read more details about Meraki webhooks, the Dashboard setup, and integration tools in the Meraki Webhooks documentation.

8.4.13

Cisco Meraki Location Scanning API

The Meraki cloud uses the physical location of access points (using the Map & Floorplan on the Dashboard) to estimate the location of a client. This should be considered a best-effort estimate, since the geo-location coordinates (latitude, longitude) and X,Y location data accuracy can vary based on a number of factors. These include AP placement, environmental conditions, and client device orientation. Experimentation can help improve the accuracy of results or determine a maximum acceptable uncertainty for data points.

Location analytics

The Location Scanning API can be used by retail stores with multiple locations, conference deployments where location information can be useful for attendees, or deployments where the business wants to know trends in user engagement. You can read about enabling the API for real-time device detection in the developer documentation.

Bluetooth Scanning API

Examples of Bluetooth Low Energy (BLE) devices include wireless headphones, wireless keyboards and mice, video game controllers, battery-powered beacons, Apple iBeacon hardware devices, fitness monitors, and remote sensors. Meraki Access Points (APs) can detect and locate these devices when they're nearby. This location data can be integrated with third-party applications.

To enable BLE device location, enable the BLE scanning radio on the access points. BLE Scanning is enabled in **Wireless › Bluetooth Settings › Scanning**.

Location and privacy

Due to the sensitive nature of location data with exact MAC addresses as part of that data, Meraki only stores a hashed version of the MAC address. It's implemented in such a way that no algorithm can recover the original MAC address of a client.

You can read details about identity protection when using Meraki and opt-in and opt-out mechanisms in the Meraki Location and Privacy documentation. Meraki's global opt-out is available at https://account.meraki.com/optout.

Meraki Smart Cameras can perform object detection, classification, and tracking directly on the edge of the network, placing the computing needs in the endpoint.

Through both REST and MQTT API endpoints, applications can request or subscribe to historical, current, or real-time data generated in the camera to use the camera for more than security. When you access the API endpoints in MV Sense, you gain access to machine learning/computer vision data for use in applications.

Camera API categories

A number of camera APIs are available for use:

- **MV Sense -** This includes REST and MQTT API endpoints, which provide historical and real-time people detection data and light readings. See MV Sense API Documentation.
- **Live Link API -** A REST API which returns a Dashboard link for a specified camera. If the link includes a timestamp, it provides data from that time. Get Network Camera Video Link API Documentation
- **Snapshot API -** A REST API that generates a snapshot of the specified camera's field of view at a specific time and returns a link to that image. Generate Network Camera Snapshot API Documentation

REST vs MQTT APIs

Two different types of API endpoints are available in the MV Sense collection of API endpoints; REST-based and MQTT-based.

RESTful APIs offer an on-demand service. A connection will be made only when data is requested. Using the MV Sense REST APIs enable historical or near real-time people detection data from the camera.

MQTT-based protocols use a publish-subscribe connection between the client and server. In the case of MV Sense, the server is continuously pushing messages to the MV smart cameras so the device can respond instantly. Using the MV Sense MQTT APIs enable a real-time feed of people detection and their locations. Light-level readings can also be obtained.

MV API use cases

Here are a few example use cases for the camera APIs:

- Detect one or several people in the vicinity of a dangerous machine, and set off a nearby warning alarm.
- Detect several people standing in one location for an extended period of time and correlate with customer wait times.
- Provide better workplace lighting by integrating MV light readings into workplace smart lighting.

You have probably experienced a "splash page" when trying to access Wi-Fi at your favorite coffee shop or in a public library. This is known as a captive portal. It is what a user sees when they first associate with a Wi-Fi SSID and open a web browser to surf the internet while in range of the Meraki device. By configuring a captive portal, you can redirect all internet traffic to a particular URL.

With that URL, you can require users to take specific actions before their traffic is able to pass through to the internet. Maybe you want to ask a customer to take a variety of actions:

- Fill out a customer survey
- Choose and purchase a billing plan
- Watch a video advertisement
- Accept a set of terms and conditions

All these actions can be required before allowing the customer to access the internet.

Read more about splash pages in the Meraki Captive Portal Configuration Guide.

8.4.14

Video - Access Meraki Dashboard Using REST APIs in Python

8.4.15

Cisco NX-OS Platform

Nexus Operating System (NX-OS) is a data center operating system for the Nexus switch.

Benefits and purpose

With the Nexus switches running the NX-OS, you can automatically provision Cisco switches in the data center and orchestrate changes much the same way you configure a Linux server.

Nexus switches are highly performant in data centers and integrate with a lot of systems. The Nexus switch family uses a Linux kernel. With this kernel, you can access a device's Bash environment and manage a switch with Linux-based commands. You can manage the device with existing management systems like Ansible or Puppet. NX-OS is interoperable with native IOS. NX-OS also provides a REST API service and model-driven programmatic access via RESTCONF, NETCONF, and OpenConfig.

Architecture

Cisco Open NX-OS leverages the native Linux networking stack, instead of a custom-built userspace stack (NetStack) that was used in prior versions of NX-OS. Virtual Routing and Forwarding actions (VRFs) are implemented using Linux network namespaces. Network namespaces provide the same isolation capabilities as VRFs.

Nexus switch interfaces, including physical, port-channel, vPC, VLAN, and other logical interfaces, are mapped to the kernel as standard Linux netdevs.

The REST API service is provided using an NGINX web server.

Environment and scale

The Nexus family of switches is used in data centers around the world. Nexus switches are used by service providers, servicing large numbers of consumers. They're performant and preferred by developers,

DevOps professionals, and other users who want to provide self-service network infrastructure models.

The Cisco Nexus 9000 Series NX-OS Verified Scalability Guide document describes the Cisco NX-OS configuration limits for Cisco Nexus 9000 Series switches.

Capabilities

NX-OS fulfills several configuration use cases, such as interface configuration, VLAN configuration, VLAN management, and Open Shortest Path First (OSPF) configuration.

Linux environment and tooling

Standard Linux tools like `ifconfig`, `ethtool`, `route`, or `tcpdump` can be used to manage a Cisco Nexus switch with NX-OS. You can also use `yum` (Yellowdog Updater, Modified) as it is the default package and repository management tool for NX-OS and has RPM underneath. These same tools can be used for NX-OS process-patching, or installing external or custom-developed programs onto the switch.

Container support

NX-OS supports running Linux Containers (LXCs) directly on the platform. It provides access to a CentOS 7-based Guest Shell, which supports custom functionality directly on the device in a secure, isolated shell.

Telemetry

You can integrate different telemetry applications such as Ganglia, Splunk, or Nagios on the switch with NX-OS.

Open NX-OS programmatic interfaces

Open NX-OS is the set of software used to provide the APIs, data models, and programmatic access. This includes the NX-API REST service and model-driven programmability using YANG modeling. Both the NX-API CLI and NX-API REST API interfaces are served by an NGINX web server. The NX-API REST model borrows several concepts from Cisco ACI and makes them available for a non-ACI based standalone Nexus fabric environment.

Open NX-OS includes the native NX-OS data model in the software image itself. If you are only using Open NX-OS, you do not have to download additional models.

YANG, NETCONF, and RESTCONF

NX-OS has a comprehensive set of both native and open YANG models, supporting Nexus switch management. The list of supported models includes native, OpenConfig, and Internet Engineering Task Force (IETF) models. Cisco NX-OS supports YANG models through the interfaces of NETCONF, RESTCONF on Open NX-OS, you must enable the features and install the desired OpenConfig models to the network switch.

The Cisco NX-OS employs a NETCONF agent as a client-facing interface to provide secure transport for the client requests and server responses. The NETCONF is enabled on Cisco Nexus 3000, 5000, 6000,

7000, and 9000 series switches.

Enabling Model Driven Programmability features in NX-OS

Note: These instructions apply to Open NX-OS 9.2.1+. Previous versions of Open NX-OS require manual installation and activation of RPMs for the protocol agents. See the Programming Guides for your platform if you are using an earlier version.

Enable the following features on the switch using CLI or another method (such as NX-API). Enable the transport protocols that you want to leverage.

```
feature bash-shell
 feature netconf
 feature restconf
```

For more information on the YANG OpenConfig models for NX-OS, see NX YANG.

OpenConfig

OpenConfig, an alternative model, must be downloaded to the switch.

OpenConfig models supported by Open NX-OS can be downloaded from Cisco Artifactory Open NX-OS Agents. You will find Open NX-OS release specific folders containing RPM packages for individual models, as well as a single RPM with all OpenConfig models.

Download the desired models to your local workstation, and then copy them to the Open NX-OS switch where you want to install them.

```
nx-osv9000-1#copy scp://developer@10.10.20.20/home/developer/Downloads/mtx-openconfig-
all-1.0.0.0-9.2.1.lib32_n9000.rpm bootflash: vrf management
```

Use the `bash-shell` feature on Open NX-OS to install the newly copied RPM files. Remember that the feature `bash-shell` must be enabled on your switch.

```
nx-osv9000-1(config)#run bash sudo su
 bash-4.2#
 bash-4.2# cd /bootflash/
 bash-4.2# yum install mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
```

8.4.16

Cisco NSO

The industry is rapidly moving towards a service-oriented approach to network management, where complex services are supported by multiple and diverse systems and processes. To manage services, operators transition from managing pieces of equipment towards actively managing the various aspects of services.

Configuring the services and the affected equipment are among the largest cost-drivers in provider networks. Still, the common configuration management practice involves pervasive manual work or ad-hoc scripting. Why do we still apply these sorts of techniques to the configuration management problem? Two primary reasons are the variations of services and the constant change of devices. These two underlying characteristics are, to some degree, blocking automated solutions, because it takes too long to update the solution to cope with daily changes.

Time-to-market requirements are critical for a new service to be deployed quickly; any delay in configuring tools impacts revenue.

In the past several years, Software-Defined Networking (SDN) has emerged as a solution. SDN provides a centralized, unified set of APIs to control both networks and services, automatically and in real time. The benefits include faster and more precise control of networks and services, leading to higher performance and lower costs.

Network Services Orchestrator (NSO) enables operators to adopt the service configuration solution dynamically, according to changes in the offered service portfolio. It delivers the SDN vision by providing a logically centralized interface to the multi-vendor network.

NSO has three components:

- Model-driven programmable interface (YANG models)
- Configuration database
- Device and service abstraction layers

Device and service abstraction layers

By using standard data models and modeling language to describe both services and devices, NSO can automate creation, deletion, and run-time modification of network services. NSO uses the standardized YANG modeling language to model and automate any type of device. The network you model can be a mix of traditional equipment and virtual devices, within Layers 1 through 7 of the OSI model.

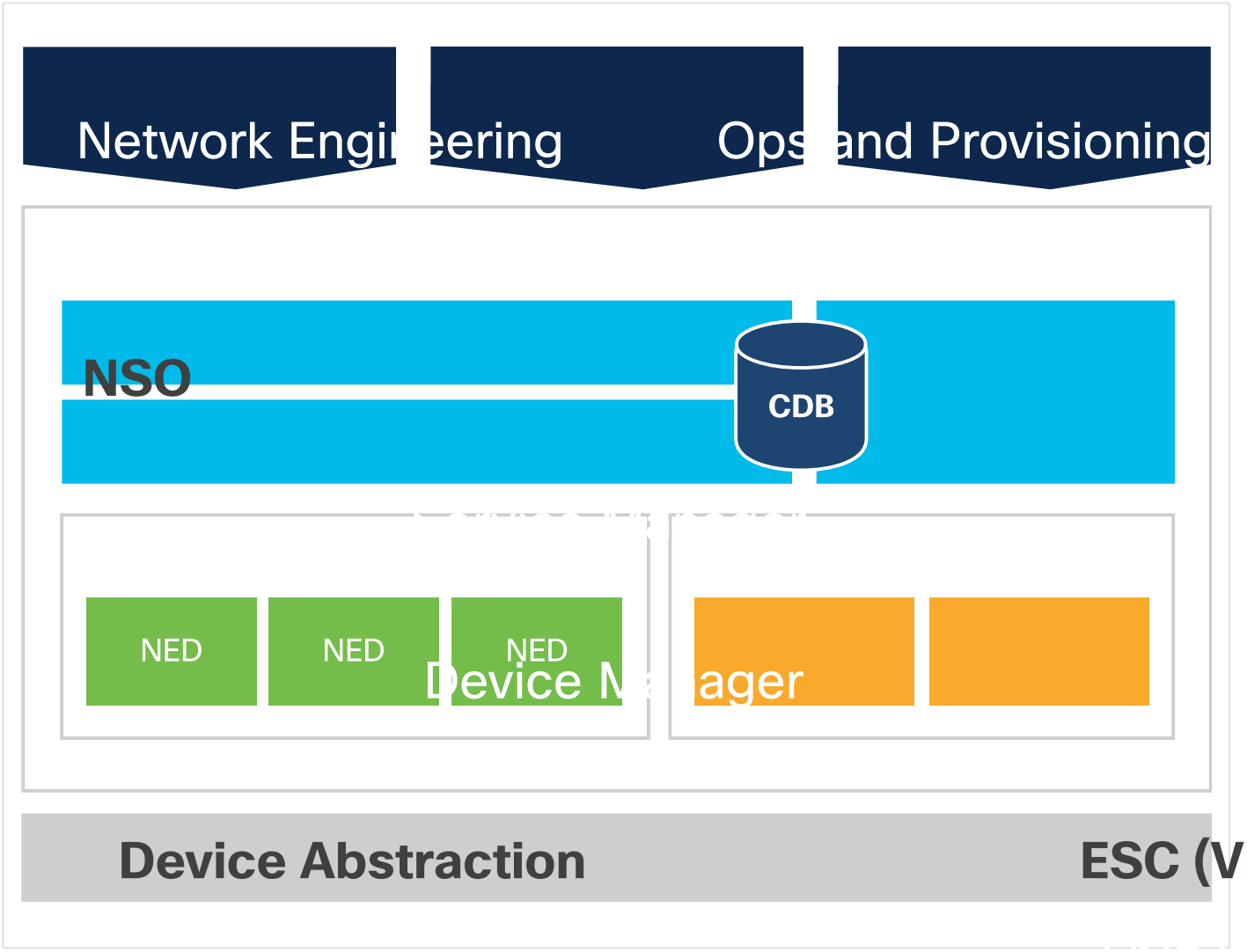
The data model for a service correlates service definitions with network operations. An embedded algorithm determines the minimum network changes required for a service, and then executes them. For device data models, NSO recognizes and can work across the physical devices in a data center, including firewalls and other OSI model Layer 4 through Layer 7 devices. It also works with virtual resources, including Virtual Machines (VMs) and container-based networking models. The Cisco Elastic Services Controller (ESC) is part of the core platform and provides these capabilities. In addition, NSO can automate the launch, configuration, monitoring, and license management of Virtual Network Functions (VNFs).

Functional architecture

The NSO architecture is logically comprised of two layers: a Device Manager that simplifies device integration and manages device configuration scenarios, and a Service Manager that applies service changes to devices. The architecture diagram below shows the required building blocks for NSO to perform network service functions in these layers. Device Manager and Service Manager components serve different purposes but are tightly integrated into one transactional engine and database.

At the core of NSO is the Configuration Database (CDB), providing persistent datastore and which synchronizes with device and service configuration. It manages relationships between services and devices and can handle revisions of device interfaces. NSO addresses the mapping, working from a desired service configuration to the corresponding device configuration and through to a dedicated mapping layer.

NSO Functional Architecture



The purpose of the Device Manager is to manage different devices using a YANG and NETCONF view. Whether the interface is SNMP or CLI, the Device Manager creates a transactional change sequence for the target devices. When the device supports YANG and NETCONF natively, the Device Manager process is automatic. Non-NETCONF devices are integrated into Device Manager using Network Element Drivers (NEDs).

The Service Manager lets you develop service-aware applications, such as switch port modification, MPLS, VPN, etc., to configure devices. Data models known as service models are created using YANG and are

based on the requirements for the service. Mapping device-to-service can be completed in two ways. For simple cases, you specify configuration templates that transform service parameters to device configuration parameters. For complex cases, you need to program "Mapping Logic" for NSO to understand. The Service Manager handles the complete lifecycle (creating, modifying, and deleting) of service instances.

Multi-domain Networks

Custom code, applications, and specific NEDs are examples of packages that NSO loads. A package consists of code, YANG modules, custom UI widgets, etc. NSO loads these at startup. Packages can be added and upgraded at run-time.

Device Manager

The Device Manager lies at the heart of NSO. The Device Manager maintains a list of all managed devices. NSO keeps the master copy of the configuration for each managed device. Whenever a configuration change is done to the list of device configuration master copies, the job of the Device Manager is to partition this "network configuration change" into the corresponding changes for the actual managed devices. Depending on the device type, different protocols are spoken southbound. If the device is or has:

- **NETCONF**-capable - Device Manager produces explicit NETCONF edit-configuration RPC operations for each participating device and then inside the same transaction that runs in NSO, executes all the individual, device-specific NETCONF operations.
- An **SNMP** device - Device Manager translates the change of the NCS DOM tree into the corresponding SNMP SET PDUs.
- A **CLI -** such as Cisco IOS or IOS XR routers (or supporting the same command structure). A CLI NED is used to produce the correct sequence of CLI commands.

Otherwise, for devices that do not fall into those categories, Java code in a Generic NED gets invoked with the proposed changes. The job for that NED code is to translate between the `diff` on the NCS DOM tree to the corresponding operations on the device.

The Device Manager supports the following overall capabilities:

- Provision a new device by copy-and-edit, either from another configuration of another device or from a template configuration.
- Deploy configuration changes to multiple devices in a fail-safe way using distributed transactions.
- Validate the integrity of configurations before deploying to the network.
- Apply configuration changes to named device groups.
- Apply templates (with variables) to named device groups.
- Roll back changes, if needed.
- Audit configurations, check if device configuration state is synchronized with the NSO CDB view.
- Synchronize the CDB and the configurations on devices.
- Connect devices to NSO using NEDs.

Service Manager

The NSO Service Manager can be used to represent the network services, such as L2 and L3 VPNs, BGP Peers, or Access Control Lists. The network services can be represented in a vendor-agnostic way, and

then NSO users can manipulate the services and let NSO calculate and apply the device changes. The objective of the Service Manager is to maintain the complete life-cycle for a network service.

A service in NSO consists of the following:

- A YANG service model - This defines the attributes of the service. For example, a Layer 2 VPN Network Service might be defined with virtual circuit ID, service identifiers, and interface names. NSO will use the YANG service model to render corresponding CLI and Web UI.
- Device configuration map - When the service is created, corresponding changes must be made to the devices. NSO supports ways to define this either with templates or with Java.

The Service Manager provides the following:

- Creating, modifying, deleting services. (FASTMAP works behind the scenes here.)
- Dry-run service life-cycle operations and report before making modifications. A dry-run report will predict changes to the devices.
- Check-sync all services or specific service. Check if the actual device configuration corresponds with the service view. This can be used to check if the device configuration has been changed out of band, or if the resulting device configuration is in violation of permitted service configurations.
- Maintaining device dependencies. Every service instance in NSO knows the corresponding device configuration. Device configurations that are the result of service provisioning can be mapped to the service instances that created the configuration.
- Service self-test. With a self-test action in NSO, you can trigger diagnostic tests of the service.

Configuration Database (CDB)

NSO uses an internal Configuration Database (CDB) to store its own configuration, the configuration of all services, as well as a copy of the configuration of all managed devices. The CDB is persistent, and all transactions are journaled to disk. CDB runs RAM-resident with the entire configuration maintained in memory, resulting in high-performance transactions. However, in tradeoff, run-time memory requirements increase with increased configuration data.

The NSO CDB provides:

- A model on how to handle configuration data in network devices, including an update mechanism upon subscription.
- An internal API for locating network element configurations.
- Automatic support for upgrade and downgrade of configuration data.

The consumers of the database services include the CLI, the web UI, and the NETCONF sessions. CDB client applications need to be able to read configuration data from the database and react when configurations are updated.

Note: The CDB journaling requires file system providing sufficient minimal performance and primitives must include file synchronization and truncation. NFS and other network file systems are unsuitable and unsupported for use with CDB in production deployment.

YANG models

NSO is model-driven with service-models expressed in YANG. Device models are also defined in YANG even though the native format may be MIBs or CLI commands. NSO is itself defined in YANG. A data model describes how data is represented and accessed. In YANG, data models are represented by definition hierarchies called schema trees. Instances of schema trees are called data trees and are encoded in XML. In NSO, there are three primary YANG sources:

- NSO data-model – This defines the built-in functions of NSO.
- Data models from devices such as native YANG modules from NETCONF devices, generated YANG modules from SNMP MIBs, or reverse engineered YANG modules from a CLI device. These YANG modules then specify the functions of the devices that are integrated to NSO.
- YANG service models – When developing service applications, a developer specifies the service model, such as a BGP peer, firewall setting, or MPLS VPN, in YANG.

These models are first compiled into NSO, which then renders all northbound APIs, User Interfaces (Web UI, CLI), and database schemas.

Service model design

A Service in NSO consists of the following:

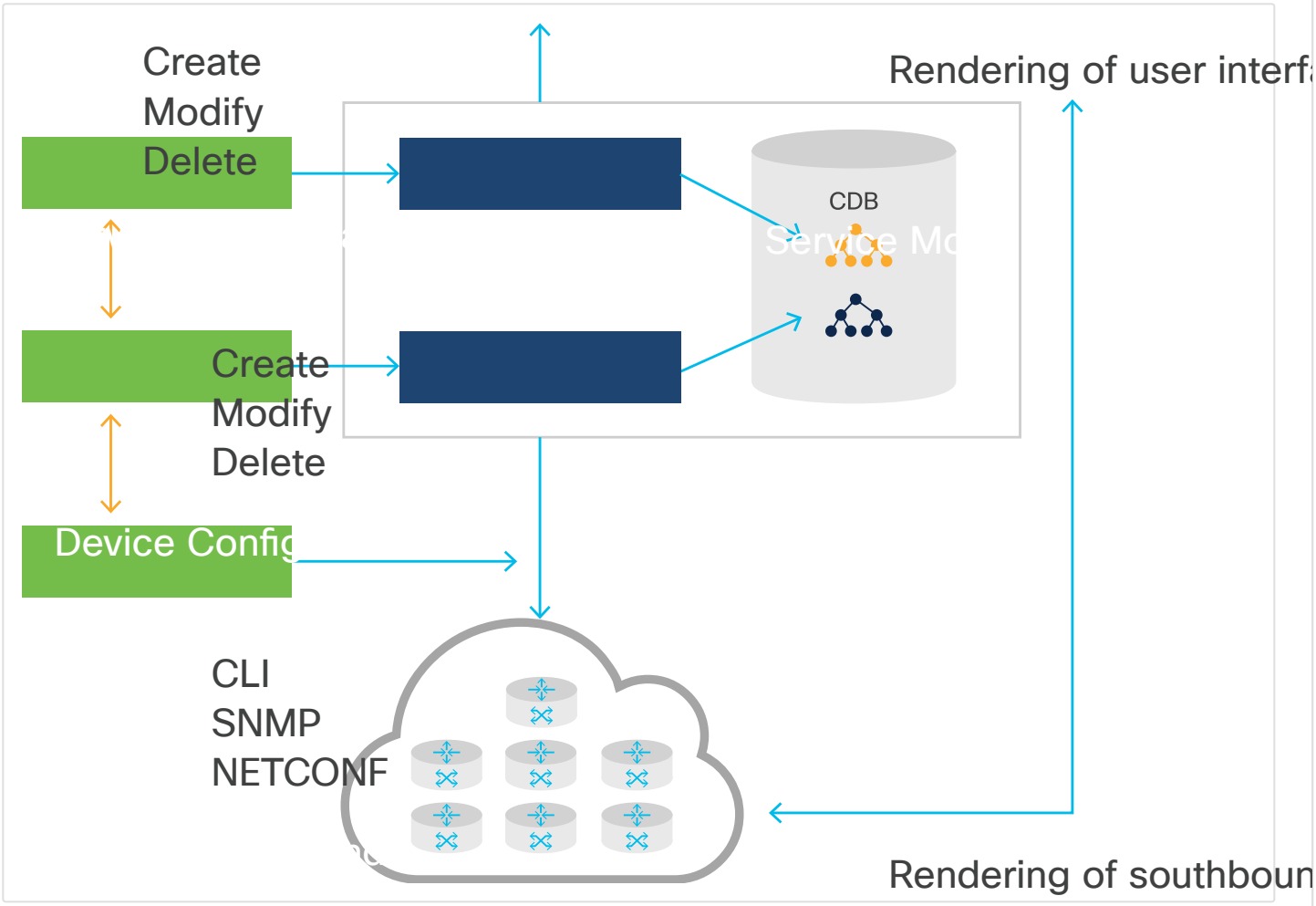
- A YANG service model – This defines the attributes of the service. For example, an L2 VPN service might be defined with virtual circuit ID, service identifiers, and interface names. A matching CLI and Web UI are then rendered to match. **Note:** While most of the validation can be expressed in YANG, in some cases, the configuration data validation will require external code, such as when performing look-ups in databases. This step is developed using MAAPI.
- A device configuration mapping – When the service is created, corresponding changes must be made to the devices. These are defined using either service templates or programmatically using Java.
  - Service templates may be used to map services to device configurations.
  - Programmatic mapping may be required for more complex mapping. It involves mapping a service operation to available device operations. (The code must handle operations such as when an access link is added to a VPN, and how this is reflected on Customer Edge and Provider Edge routers.)

The core technology of NSO is the capability to read and write configuration to and from the devices. NSO stores the service instances and the device configurations in the Configuration Database (CDB). In contrast to "offline" inventory systems, NSO focuses on transactional integrity between the database and the network, resulting in a real-time database that is always in synchronization with the network. Therefore, NSO can quickly compare a new desired device configuration and to the current configuration. NSO can then automatically render the underlying device commands that will be required to change to the new desired configuration.

The transformation from service models to device models can be complex, especially when dealing with actual configuration details. The FASTMAP algorithm, used by NSO, greatly simplifies this task. Typically, an engineer defines a higher-level service model representing actual device configurations. FASTMAP enables automatic management of any kind of change or deletion. All change scenarios are inferred from a single definition of the service. The mapping from the service create to the corresponding device configuration is defined in a high-level API or through a template. The programmer has to only define the service create method. If a user changes or deletes an existing service later on, NSO calculates the changes.

NSO service mapping is illustrated below.

NSO Service Mapping



8.4.17

Cisco NSO Services

An NSO service is a function provided by network devices. Creating, modifying, or deleting the service manipulates the actual configuration of the end devices. However, an NSO service does not actually include manual steps. Instead, service transactions performed complete logical operations meeting ACID (Atomic, Consistent DB, Isolated, and Durable) transaction properties. That is, the transaction must either complete as a unit, or not execute at all, and the database must be maintained in a consistent state.

A service model defines the required inputs in order to deploy the service in the network. NSO collapses the service logic problem to a YANG-to-YANG model transformation. If the mapping is straightforward, this can be expressed as a device template. If it is more complex, it can be expressed in a programmatic way using Java or Python. Only 'service creation' needs to be defined. The FASTMAP engine automatically manages all other service life-cycle changes such as modify, add, and delete.

Given a service model, the mapping logic reads the service model instance and updates the corresponding device tree. Mapping logic can be interface-agnostic and unaware of interface specifics such as if it is handled through CLI, NETCONF, or SNMP. The mapping logic also does not include error handling for southbound interface errors which is managed automatically by the transaction manager.

Northbound interfaces

Northbound interfaces allow integration of NSO with applications and portals. NSO has a set of northbound interfaces, including human interfaces like web UI and a CLI; programmable interfaces including RESTCONF, NETCONF, and language bindings including Java, Python, and Erlang. The list below outlines the available Northbound interfaces:

- **CLI** - The Command Line Interface is one of the flagships of NSO. It is a CLI with features such as tab completion, command history, and model awareness. This is a network-wide CLI that provides transactions across multiple devices and full support for the service models. This enables network automation by scripting towards the network and the network services in a unified CLI rather than scripting towards the devices and native CLIs.
- **RESTCONF API** - The RESTCONF API is a standardized REST interface as defined in RFC 8040 and provides a number of improvements over the proprietary and legacy REST interface, including the support for auto-generating Swagger/OpenAPI documents from YANG.
- **NETCONF** - This gives other OSS applications a NETCONF interface to the devices and services managed by NSO.
- **Java -** The Java interface is used to build applications and clients for NSO. The Java interface is model-aware; programmers can work with classes from the service and device data models. This eases the programming learning curve and assures program correctness at compile time.
- **JavaScript** - The JavaScript interface is used to build custom Web interfaces. The interface is AJAX-based so that true asynchronous clients can be created.
- **SNMP** - NSO can present operational data over SNMP to upper layer management systems. This can be useful for handling alarms and gathering performance data. There is also a dedicated SNMP Alarm MIB for the NSO Alarm Manager.
- **Web UI** - In the same sense as the CLI, the Web UI is generated dynamically from the models. The NSO Web UI can be highly customized.
- **MAAPI** - MAAPI comes in several flavors. It is the general Management Agent API. MAAPI is available as command line utilities: C-API, Java API, and a Python API.
- **REST API** - The REST interface provides a complete model via HTTP primitives. Exchanged data may be XML and JSON. **Note:** The legacy REST API has been deprecated since NSO 5.1 and is scheduled to be removed in NSO 5.3.

Command line interface (CLI)

The NSO CLI provides a unified CLI towards the complete network. It comes in two flavors: Juniper-style and Cisco XR-style. NSO CLI is a single interface for network devices and network services. Note that this is different than a "cut-through" CLI that reaches the devices directly.

As with many CLIs, there are operational and configuration modes. Depending on the mode, a **show** command will use different data sets. In configuration mode, it displays network configuration data from the NSO CDB configuration store. In operational mode, it will use live values from the devices and plus

operational data stored in CDB. The CLI starts in operational mode. Different prompts are used for different modes helping to distinguish, when used interactively.

The CLI is automatically rendered using the data models described by the YANG files. There are four distinct types of YANG models: the built-in NSO YANG models for the device manager, the service manager models, YANG models imported from the managed devices, and service models. Regardless of model type, the NSO CLI seamlessly handles all models as a whole to produce auto-generated CLI. Auto-generated CLI supports:

- Unified CLI across network, devices, and network services
- Command line history and command line editor
- Tab completion for content of the configuration database
- Monitoring and inspecting log files
- Inspecting the system configuration and system state
- Copying and comparing configuration between different parts of the CLI, such as between two interfaces or two devices
- Configuring common setting across a range of devices

Web user interface

The NSO Web UI is also automatically rendered from the YANG data models. It mirrors the data model for NSO itself, as well as the data models for the managed devices and services. As soon as the models are updated, or new devices or services are added, the UI is also updated.

The Web UI is a YANG model browser with additional device and service functionality. The interface is built with pure client-side JavaScript. The Web UI is a mix of custom-built widgets and auto-rendering of the underlying device and service models. All major web browsers are supported, and no plugins are required. The Web UI is available on port 8080 on the NSO server. The port number is configured in `ncs.conf`.

Managing services (southbound)

NSO requires a YANG model, device address, management port, and authentication credentials for each device to be managed. YANG models are imported using the NSO YANG compiler.

A name identifies each managed device. Most commonly, this is the DNS name or IP address of the device but can be any text string value. Each managed device has a mandatory IP/port pair that, together with the authgroup leaf, provide information as how to connect and authenticate over SSH/NETCONF or TELNET. The device-type determines the communication method or protocol used to communicate with the device. Those are detailed under **Southbound interfaces**, below.

Southbound interfaces

Southbound interfaces allow the configuration and management of network elements. Available Southbound interfaces:

- **NETCONF** - (default) NSO automatically discovers and uses devices supporting NETCONF.
- **SNMP** - NSO can be configured to use device SNMP if provided with MIBs and additional declarative information.

- **CLI** – A corresponding CLI NED is required. Devices supporting the Cisco CLI engine are supported.
- **IOS, IOS XR** – A corresponding NED is required and must be loaded.
- **Other/generic** – A corresponding NED, YANG models, and Java code, is required.

Adding devices

A new device can be added into NSO using any of the following methods:

- Discovery
- Manually
- Cloning
- Templates

Example CLI sequence to manually add a device:

```
ncs(config)# devices device ce9 address 127.0.0.1 port 10031
ncs(config-device-ce9)# device-type cli ned-id cisco-ios
ncs(config-device-ce9)# authgroup default
ncs(config-device-ce9)# commit
```

NSO provides the ability to synchronize device configuration between the device and NSO. In the normal case, the configuration on the device and the copy of the configuration inside NSO should be identical. You can force direction so that either NSO or the device has precedence.

NSO device `sync-from` example:

```
ncs(config)# devices sync-from
sync-result {
device ce0
result true
}
sync-result {
device ce1
result true
}
```

Configuring devices

You can also configure several devices inside the same network transaction. First, initiate `ncs (config)` mode.

Multi-device simultaneous configuration example:

```
$ ncs_cli -C -u admin
ncs# config
```

Continue in `ncs (config)` mode:

```
ncs(config)# devices device pe1 config cisco-ios-xr:snmp-server community public RO
ncs(config-config)# top
ncs(config)# devices device ce0 config ios:snmp-server community public RO
ncs(config-config)# devices device pe2 config junos:configuration snmp community
public view RO
ncs(config-community-public)# top
ncs(config)# show configuration
devices device ce0
config
ios:snmp-server community public RO
!
!
devices device pe1
config
cisco-ios-xr:snmp-server community public RO
!
!
devices device pe2
config
! first
junos:configuration snmp community public
view RO
!
!
!
```

8.4.18

Video – Access Cisco NSO Using REST APIs in Python

8.4.19

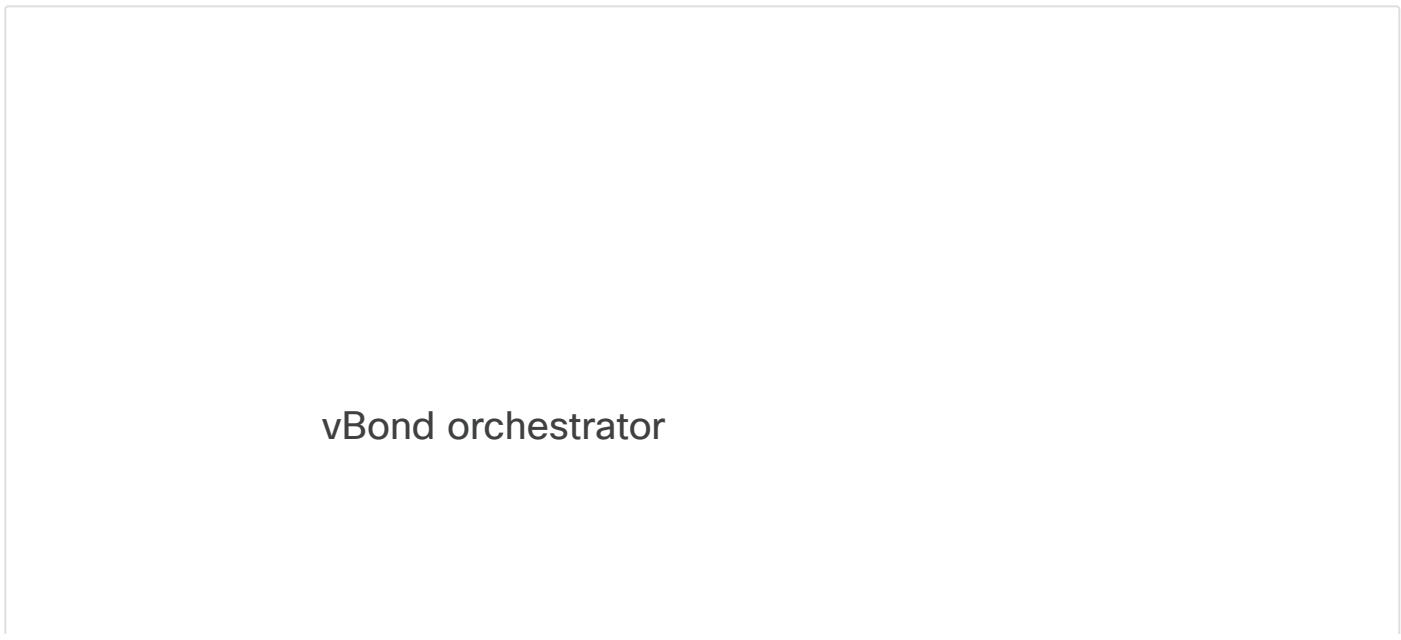
Cisco SD-WAN

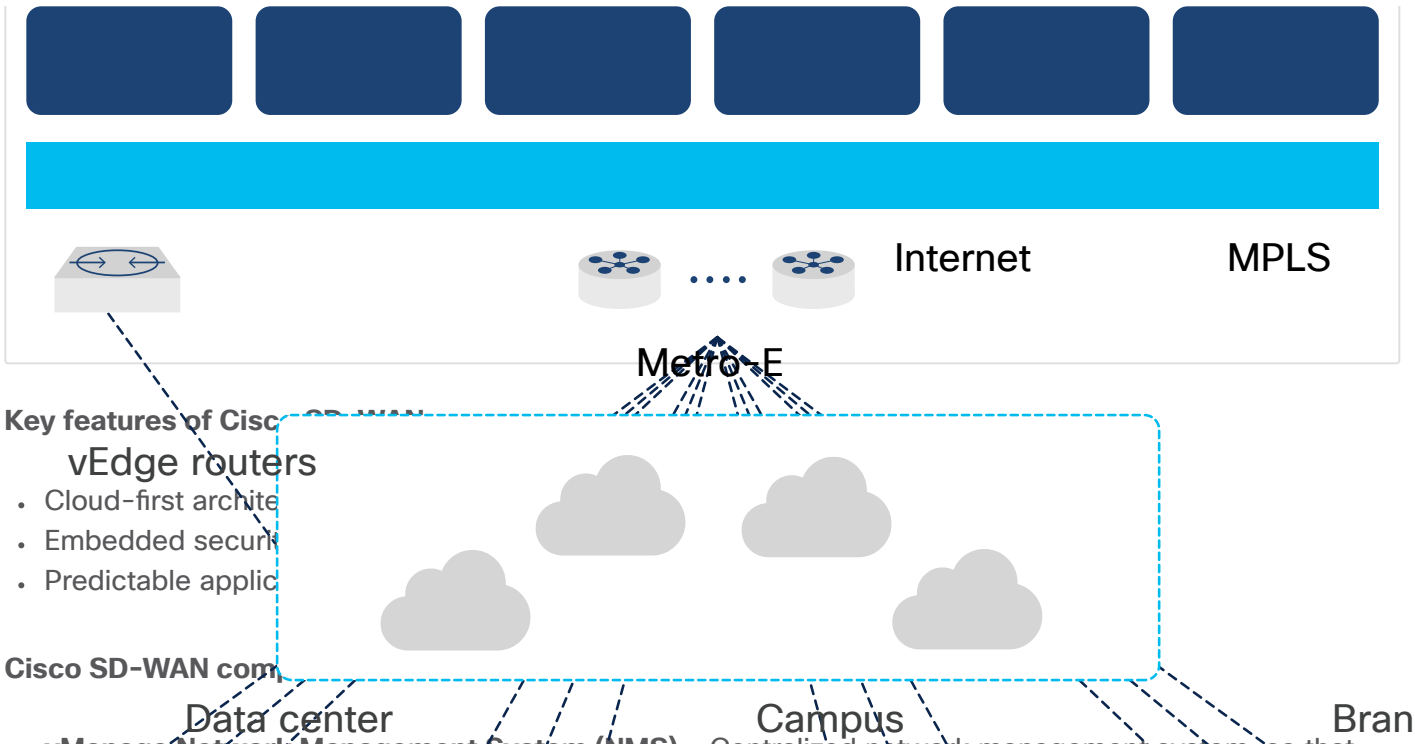
Cisco Software-Defined - Wide Area Networking (SD-WAN) supports third-party API integration, allowing for simplicity, customization, and automation in day-to-day operations. Cisco SD-WAN includes the common protocols used for all enterprise SD-WAN deployments, such as Border Gateway Protocol (BGP), Open Shortest Path First (OSPF), Virtual Router Redundancy Protocol (VRRP), and Internet Protocol version 6 (IPv6).

Through a dashboard called vManage, Cisco SD-WAN provides:

- **Transport independence** - Guaranteeing zero network downtime, Cisco SD-WAN automates application flexibility over multiple connections, such as the internet, MPLS, and wireless 4G LTE.
- **Network services** - Deliver rich networking and security services with clicks on the dashboard. WAN optimization, cloud security, firewalls, IPS, and URL filtering can be deployed wherever needed across the SD-WAN fabric from a single location.
- **Endpoint flexibility** - Cisco SD-WAN can simplify connectivity across branches, campuses, data centers, or cloud environments, extending the SD-WAN fabric wherever you need it to go.

SD-WAN Overview





Key features of Cisc

- Cloud-first archite
- Embedded securi
- Predictable applic

Cisco SD-WAN com

- **vManage Network Management System (NMS)** - Centralized network management system, so that you can configure overlay networks from a dashboard.
- **vSmart Controller** - Controls the flow of data traffic by working with the vBond orchestrator to authenticate SD-WAN devices as they join the network. It also orchestrates connectivity among the vEdge routers.
- **vBond Orchestrator** - Orchestrates connectivity between vEdge routers and vSmart controllers. If any vEdge router or vSmart controller is behind a NAT, the vBond orchestrator also serves as an initial NAT-traversal orchestrator.
- **vEdge Routers** - Provisioned at the perimeter of a site (such as remote offices, branches, campuses, data centers), and delivered as hardware, software, cloud or virtualized components, vEdge Routers secure virtual overlay network over a mix of WAN transports.

vManage management plane

The vManage NMS is a centralized network management system. The vManage NMS dashboard provides a visual window into the network, and it allows you to configure and manage SD-WAN network devices. The vManage NMS software runs on a server in the network. This server is typically situated in a centralized location, such as a data center. It is possible for the vManage NMS software to run on the same physical server as vSmart controller software.

vSmart control plane

The vSmart control plane maintains a centralized route table that stores the route information, called OMP routes. It learns the route information from the vEdge routers and from any other vSmart controllers in the SD-WAN overlay network. Based on the configured policy, the vSmart controller shares this route information with the SD-WAN network devices in the network so that they can communicate with each other.

vBond orchestration plane

The vBond orchestrator coordinates the initial start-up of vSmart controllers and vEdge routers, and it facilities connectivity between vSmart controllers and vEdge routers. During the start-up processes, the vBond orchestrator authenticates and validates the devices wishing to join the overlay network. This automatic orchestration process prevents errors from manual starts.

vEdge data plane

The vEdge router, whether a hardware or software device, is responsible for the data traffic sent across the network. When you place a vEdge router into an existing network, it appears as a standard router. Envision a vEdge router and an existing router that are connected by a standard Ethernet interface. These two routers appear to each other to be Layer 3 endpoints, and if routing is needed between the two devices, OSPF or BGP can be enabled over the interface. Standard router functions, such as VLAN tagging, QoS, ACLs, and route policies, are also available on this interface.

Cisco SD-WAN APIs

The Cisco SD-WAN software provides a REST API, which is a programmatic interface for controlling, configuring, and monitoring the devices in an overlay network. You access the REST API through the vManage web server.

You can access the API documentation in the vManage REST APIs Command Reference.

The Cisco SD-WAN REST API calls expose the functionality of the software and hardware features and of the normal operations you perform to maintain SD-WAN devices and the overlay network itself. In REST API terminology, each of these features or operations is called a resource. A resource is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

Resources are grouped into collections. Each collection contains a single type of resource, and so is homogeneous. In the REST API, the collection of resources is present at the top level of the API. The SD-WAN REST API resources are grouped into the following collections:

- Administration
- Certificate Management
- Configuration
- Device Inventory
- Monitoring
- Real-Time Monitoring
- Troubleshooting Tools

Exploring the API with vManage API Explorer

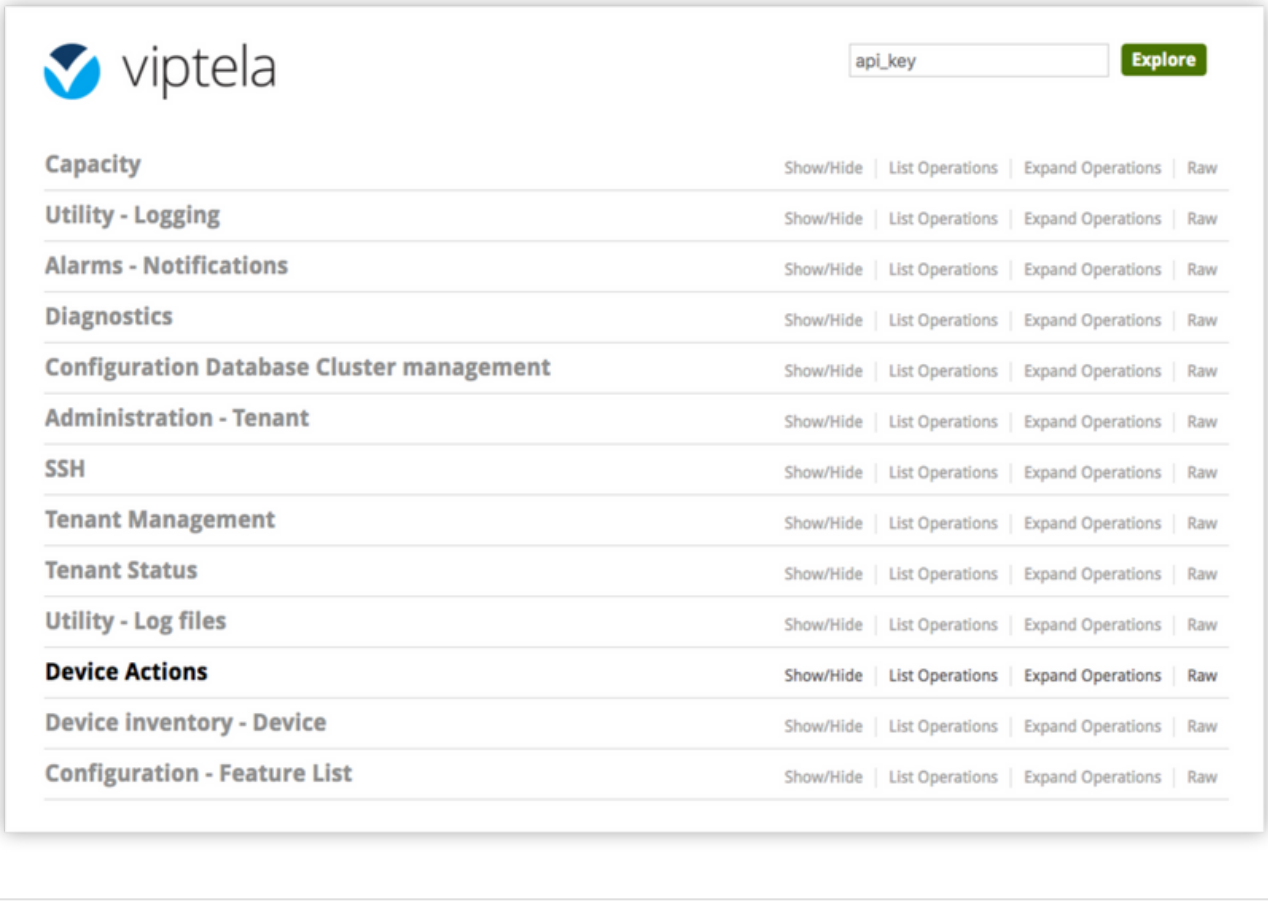
You can explore the API documentation and even try it out by logging into the `apidocs` resource on your vManage platform as: `https://<vmanage_host>:<vmanage_port>/apidocs`

Try it: https://devasc-sdwan-1.cisco.com/apidocs/

Log in using username `devnetuser` and password `RE!_Yw519_27`.

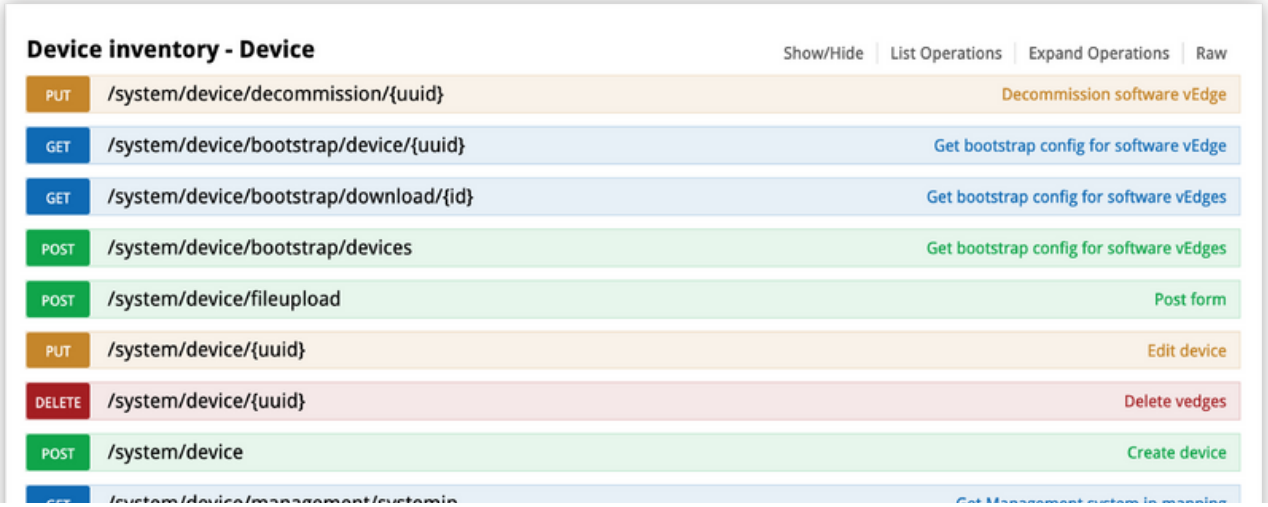
After you are logged in, you will see a list of API functional groups.

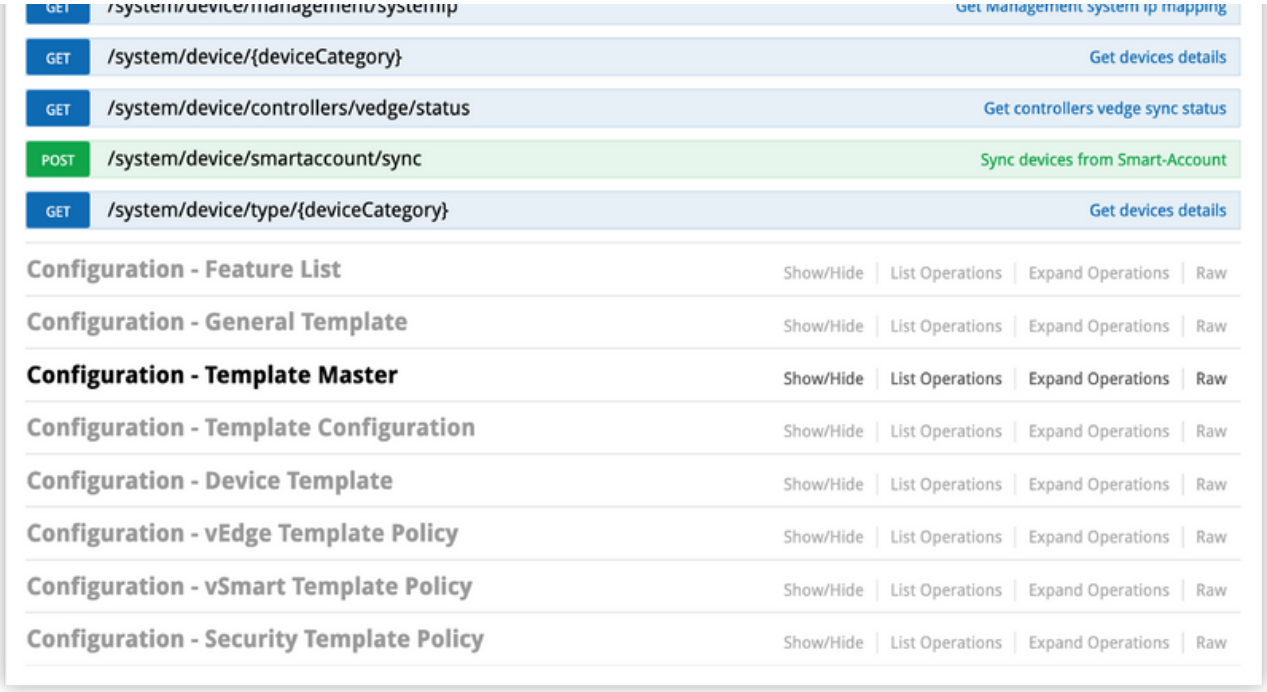
SD-WAN vManage APIs



Click any API group name, and the display will expand to a list of APIs within that functional group. Shown below are the various API calls available under the Device Inventory category. Note that it supplies the required method (GET, PUT, POST, DELETE) as well as the required endpoint URL for each API call.

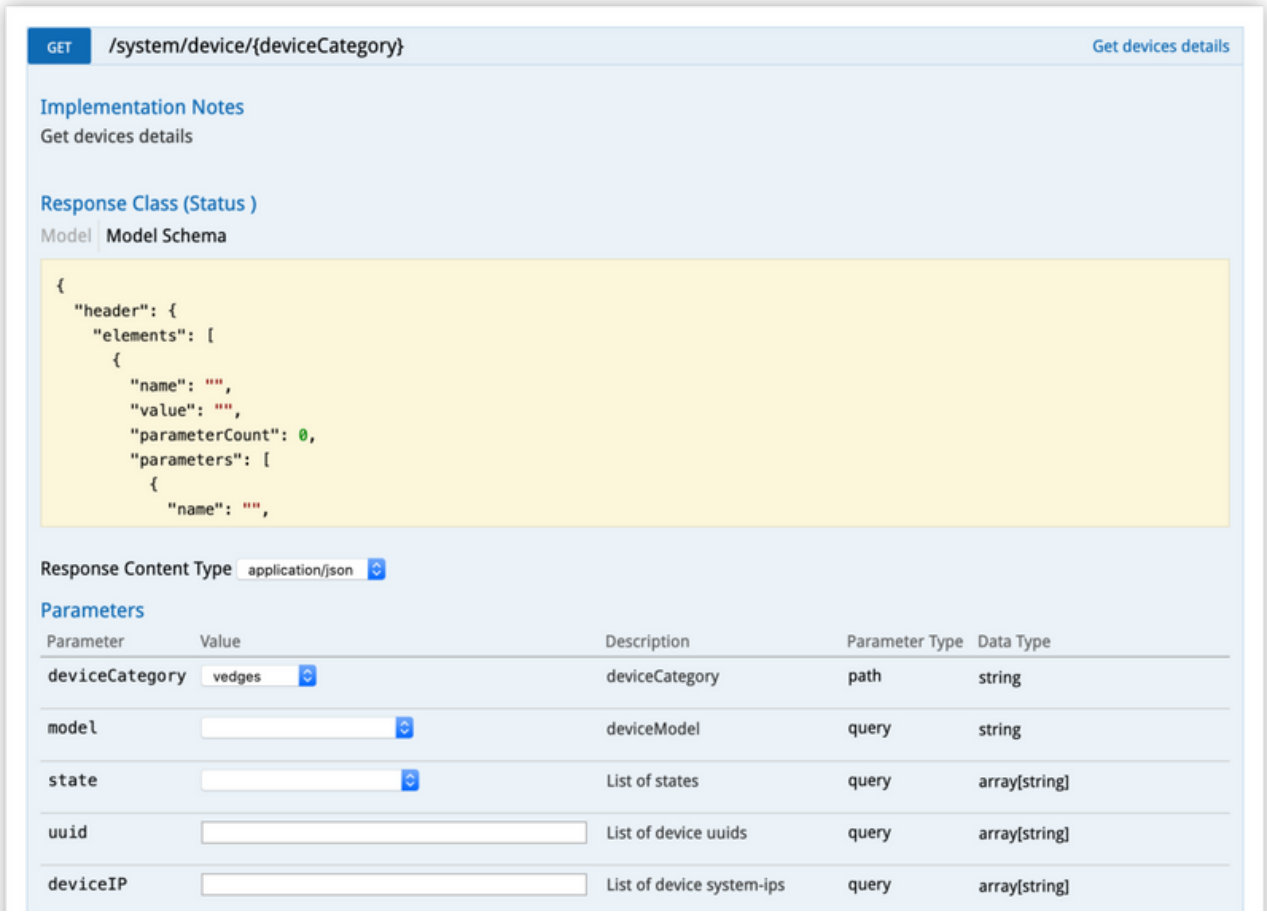
Device Inventory APIs

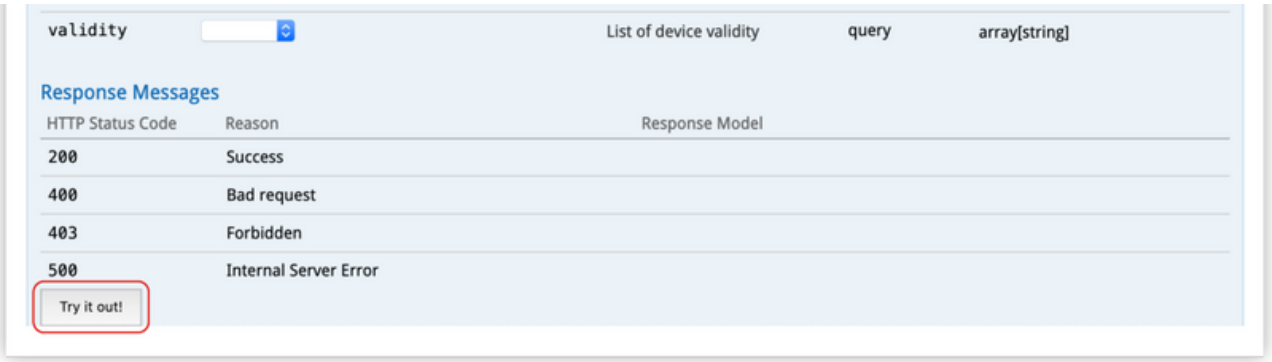




Selecting `GET /system/device/{deviceCategory}` will show detailed information on how to use the API to get a list of devices for that category.

Lists the Device Inventory Details using GET Method





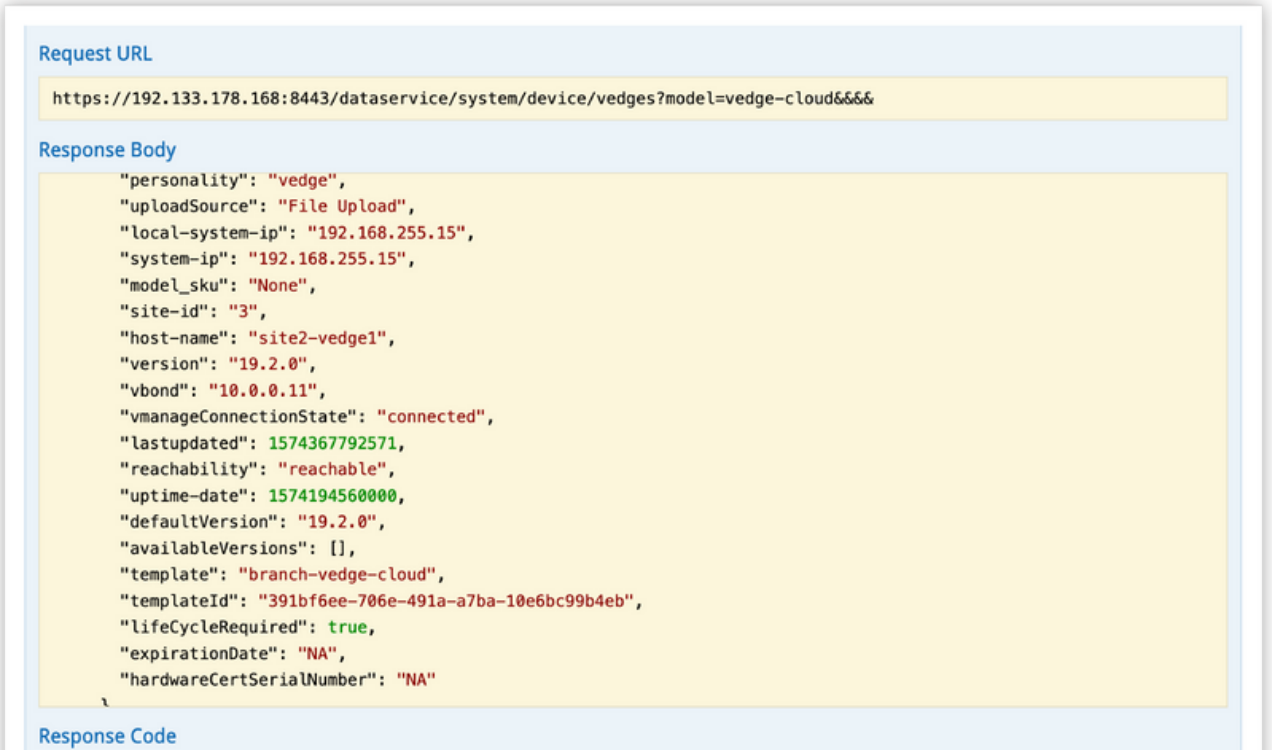
The important areas to note about this view are:

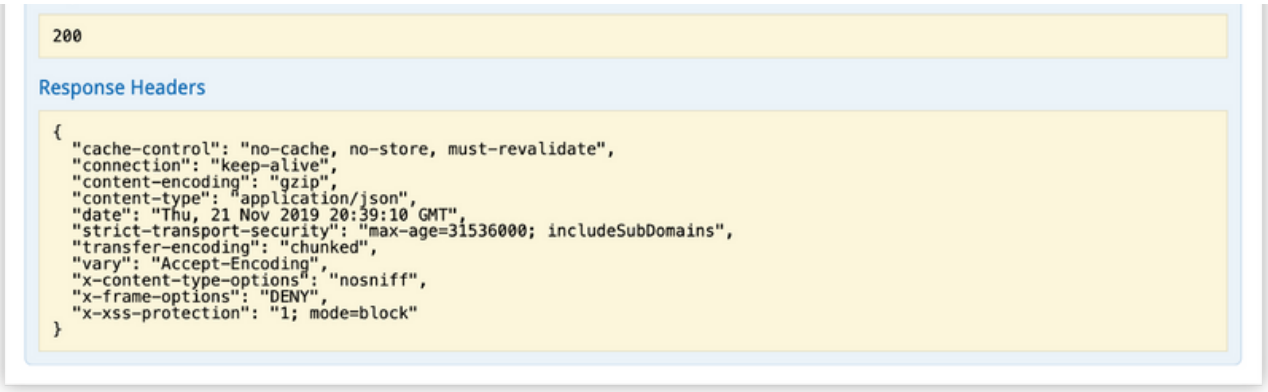
- **Response Content Type –** In this case, the response will be in JSON format, so you need to be prepared to parse JSON.
- **Parameters –** These are the parameters you can pass into the API call to filter the returned data.
- **Response Messages –** The list of HTTP codes that can be returned.
- `Try it out!` – This is the activation button at the bottom of the page.

Set the `deviceCategory` parameter to `vedges` and the `model` parameter to `vedge-cloud`, then click `Try it out!`. Clicking this button executes the API call and passes in the parameters to filter the results to only vEdge Cloud models in the device inventory.

Note: Authentication is not required when using "Try it out!" from vManage, because you are already logged in to the vManage UI and have appropriate privileges.

GET Device API Example





Areas to note about this view:

- **Request URL -** This is the URL used to issue the request. The host name displayed in this URL is the IP address of this vManage host.
- **Response Body -** In this case, note the `host-name`, `system-ip`, and `version`.
- **Response Code -** In this case, we received 200, which means Success.

8.4.20

Video - Access Cisco SD-WAN Using REST APIs in Python

8.3
Understanding Network Programmability an...

8.5
Cisco Compute Management