

Examining Cisco Integration with Automation and Orchestration Software Platforms



Sean Douglas

DATA CENTER ENGINEER

@ocdlearning



Overview



Cisco NX-OS supports intent-based automation through various tools

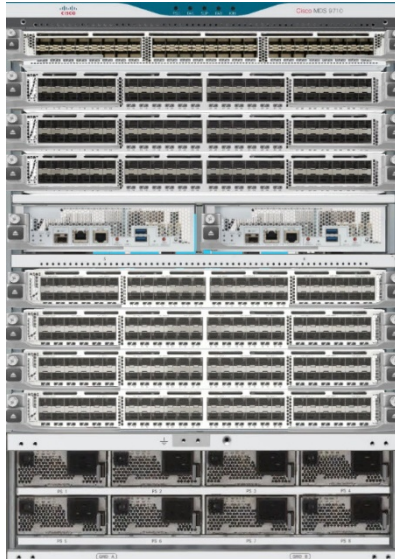
- NX-API
- NX-API REST
- Ansible
- Puppet



Automation Using Cisco Nexus API



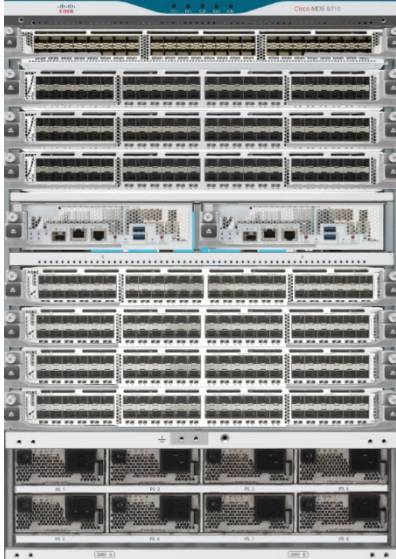
Why Automate?



Save Time



Why Automate?



Save Time

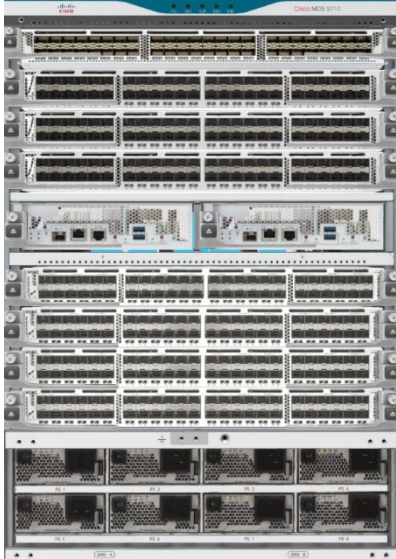
```
$ curl -u user:admin:cisco123 -X POST -d @api.xml -H "Content-type: text/xml" http://192.168.1.2/ins
<ins_api>
  <type>cli_show</type>
  <version>1.2</version>
  <sid>0</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show hostname</input>
      <msg>success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>

<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>0</sid>
  <input>show hostname</input>
  <output_format>xml</output_format>
</ins_api>
```

Minimize Human
Error



Why Automate?



Save Time

```
$ curl -u user:admin:cisco123 -X POST -d @api.xml -H "Content-type: text/xml" http://192.168.1.2/ins
<ins_api>
  <type>cli_show</type>
  <version>1.2</version>
  <sid>0</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show hostname</input>
      <msg>success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>

<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>0</sid>
  <input>show hostname</input>
  <output_format>xml</output_format>
</ins_api>
```

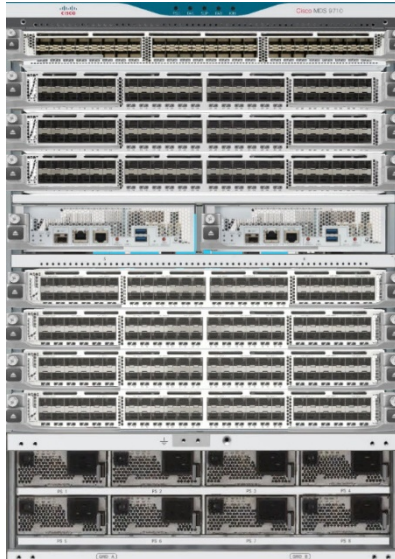
Minimize Human
Error



Customization



Why Automate?



Save Time

```
curl -u user:admin:cisco123 -X POST -d @api.xml -H "Content-type: text/xml" http://192.168.1.2/ins
<ins_api>
  <type>cli_show</type>
  <version>1.2</version>
  <sid>esc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show hostname</input>
      <msg>success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>

<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>show hostname</input>
  <output_format>xml</output_format>
</ins_api>
```

Minimize Human
Error



Customization

Create Service Profile in org root

Boot Order

Primary Boot Device ☒

Secondary Boot Device

Type:

- ☐ Local Disk
- ☐ Local LUN
- ☐ SD Card
- ☐ Internal USB
- ☐ External USB
- ☒ SAN
- ☐ LAN
- ☐ CD/DVD
- ☐ Local CD/DVD
- ☐ Remote CD/DVD
- ☐ Floppy
- ☐ Local Floppy
- ☐ Remote Floppy
- ☐ Remote Virtual Drive

SAN

vhba:

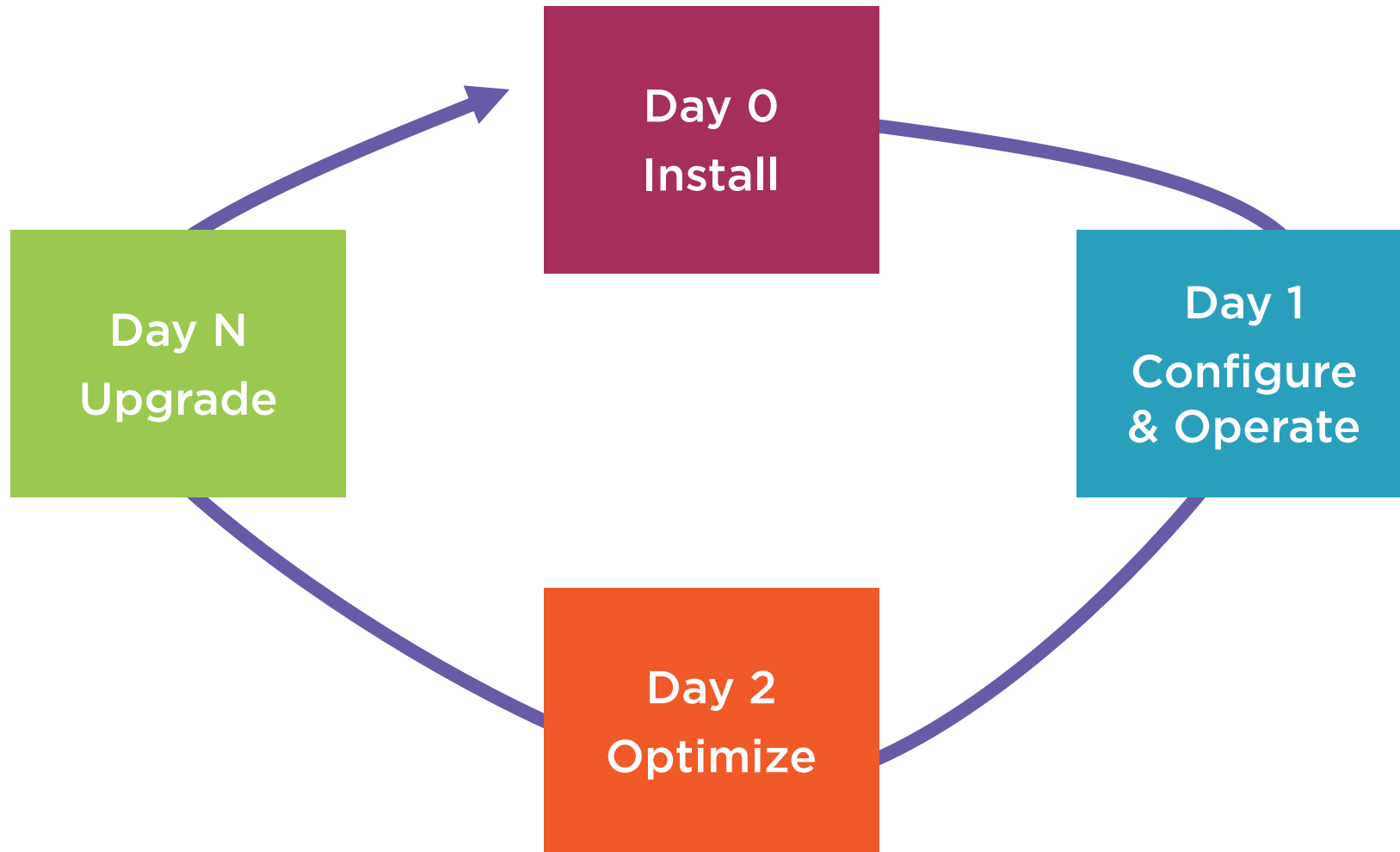
LUN:

WWN:

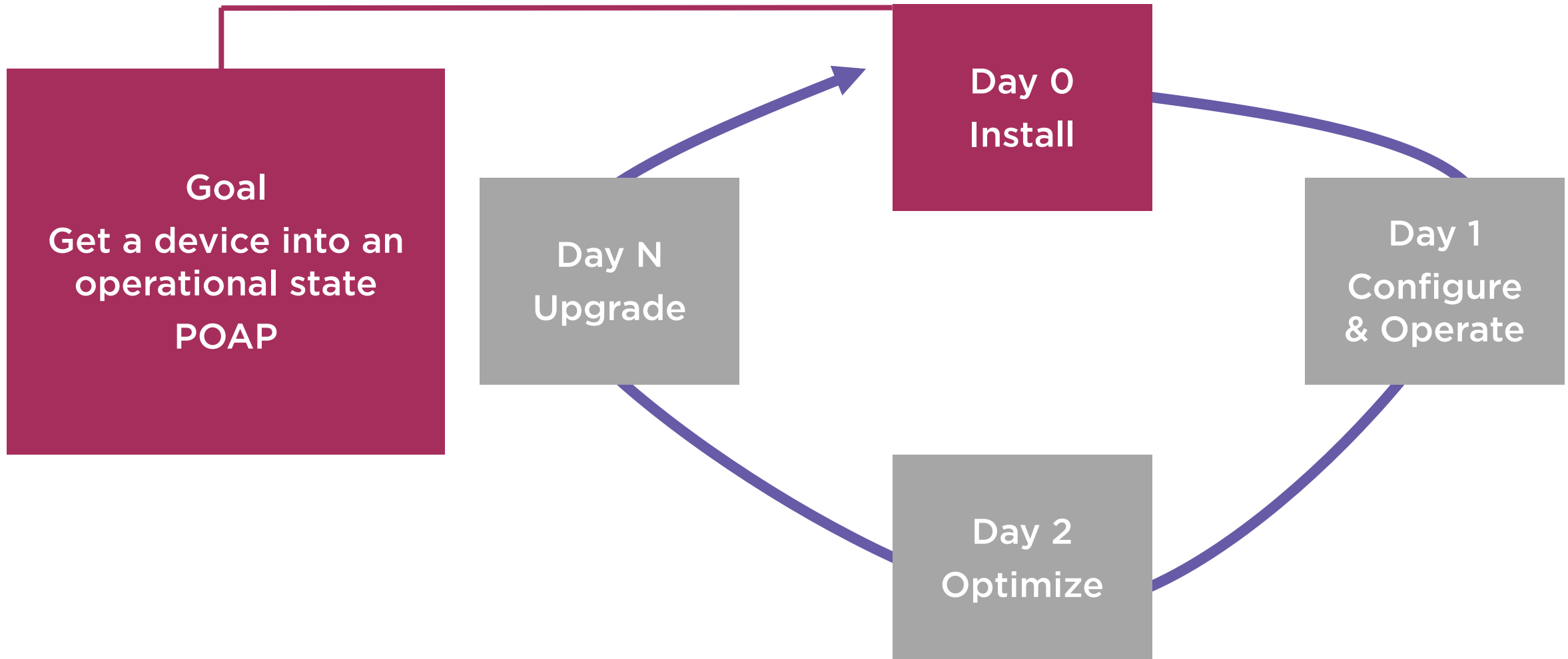
Innovation



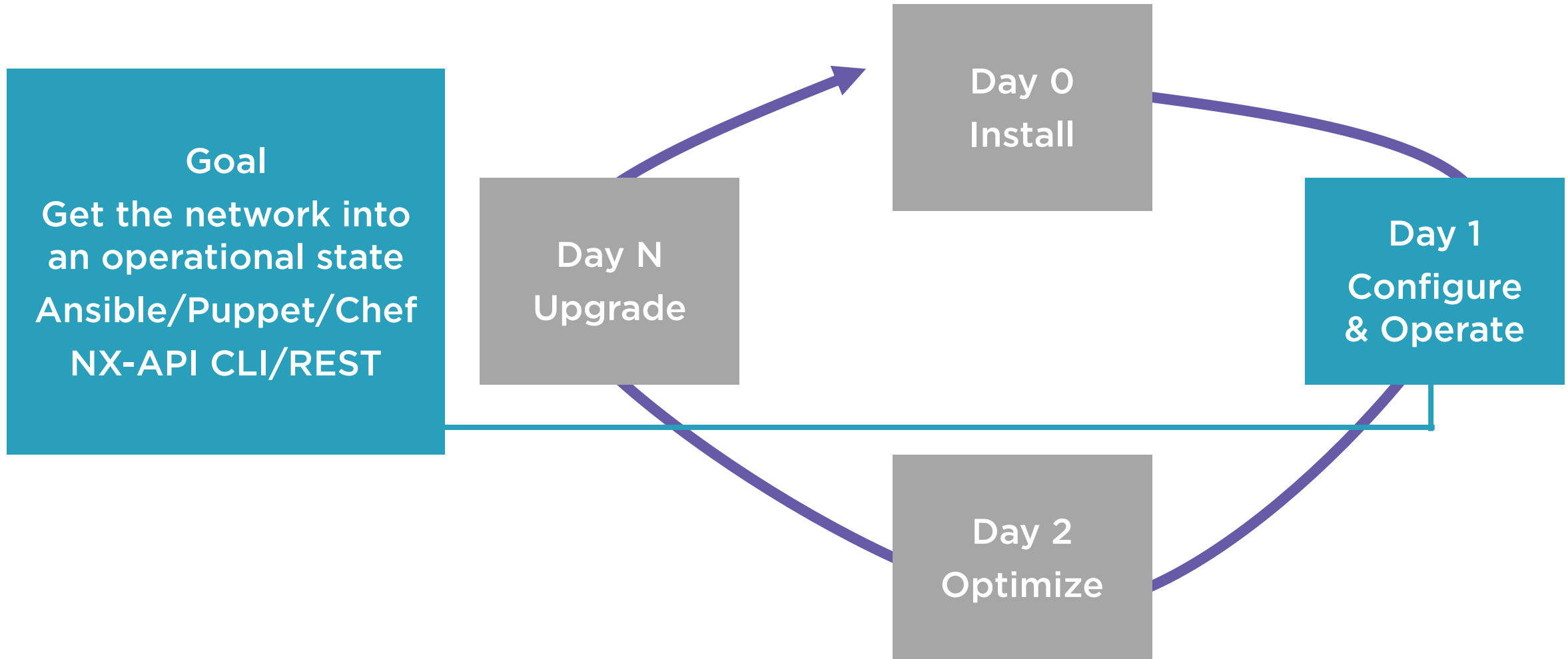
Automating Device Operational Lifecycle



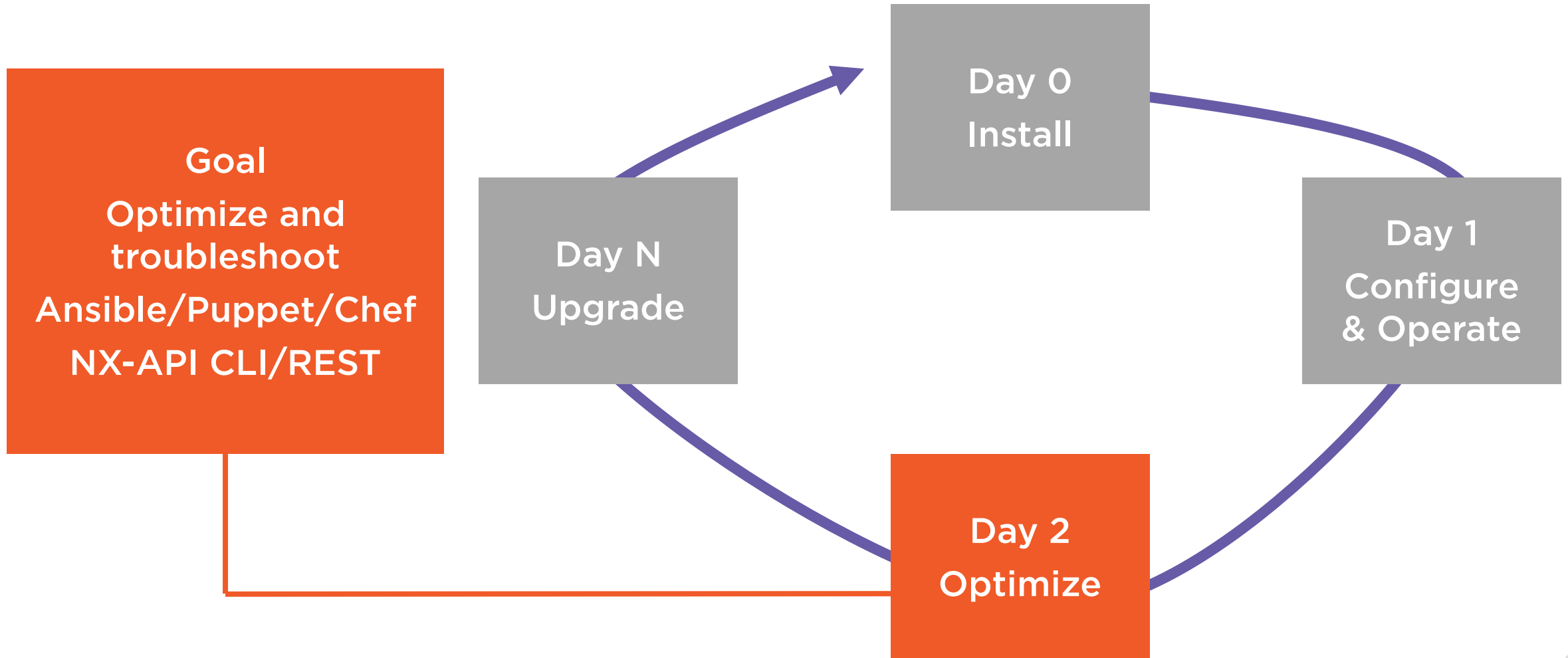
Automating Device Operational Lifecycle



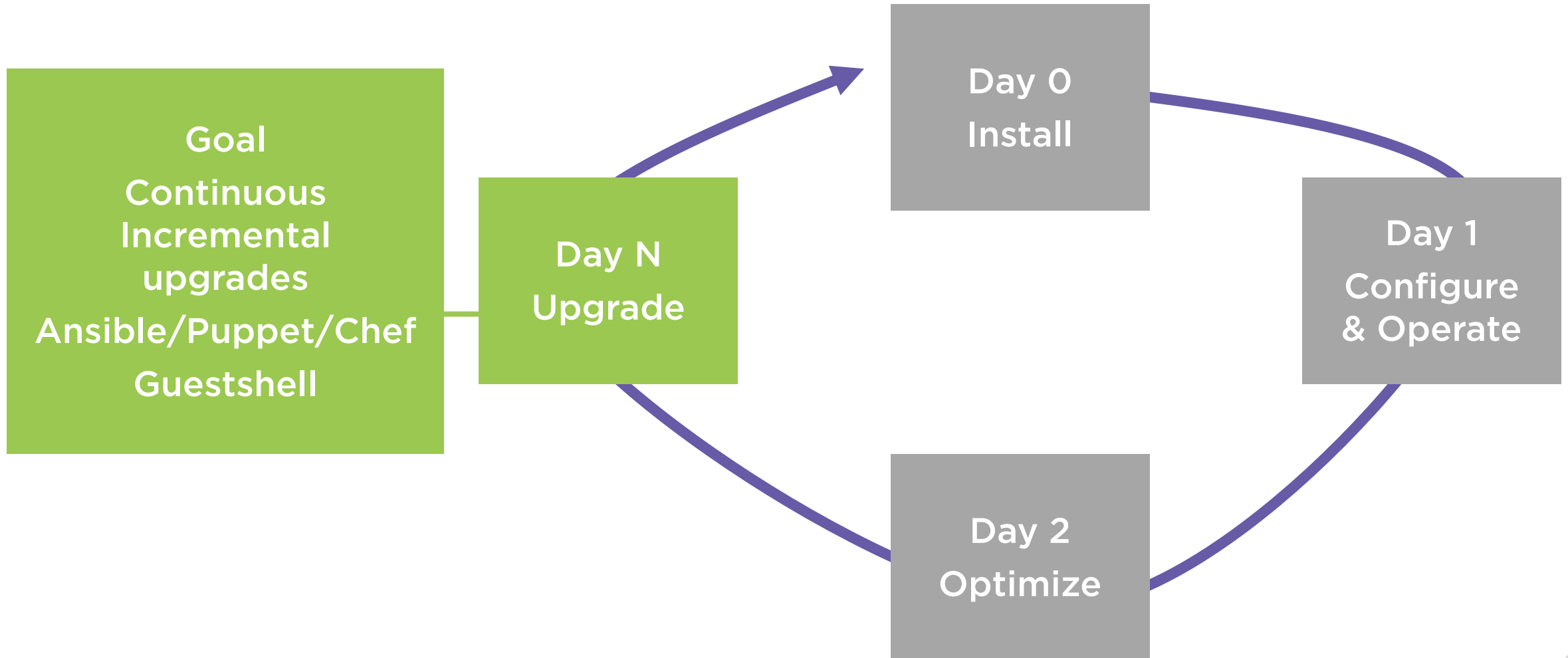
Automating Device Operational Lifecycle



Automating Device Operational Lifecycle



Automating Device Operational Lifecycle



NX-API Features



Web Base (HTTP/S)

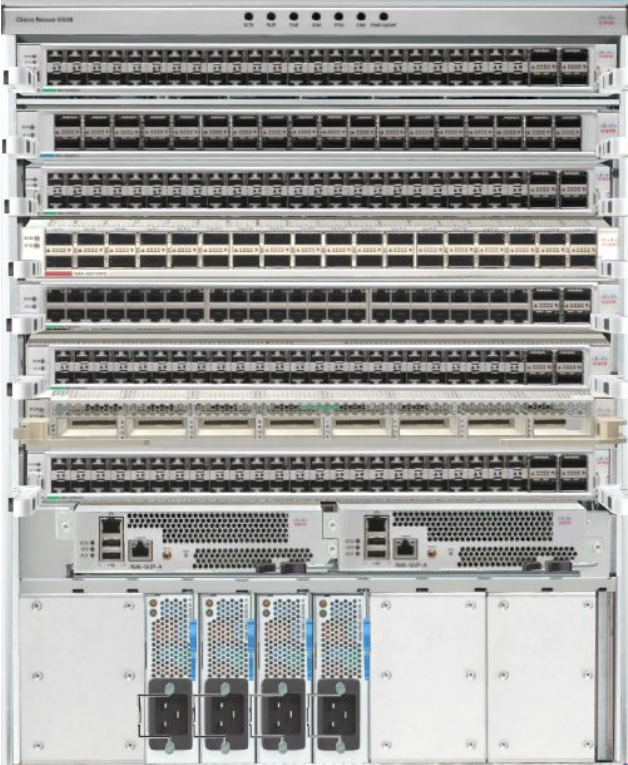


Structured Output
JSON, XML, JSON-
RPC



Role Based Access

Cisco NX-API Characteristics



Application programming interface is way for two systems to talk to each other

Provides programable access to the switches over http or https

Allows us to remotely issue commands and receive responses

NX-API uses format in structures

- XML
- JSON
- JSON-RPC

NX-API CLI Uses

A set of function calls that allow talking to the system

Check version of multiple switches at once

- Provision VLANs
- Get RIB from the switch and check flapping by checking the age of routes



```
switch# configure terminal  
switch(config)# feature nxapi
```

Enable NX-API

Before we can use the NX-API, we must enable the API




```
switch(config)# nxapi http port 8080
```

```
!
```

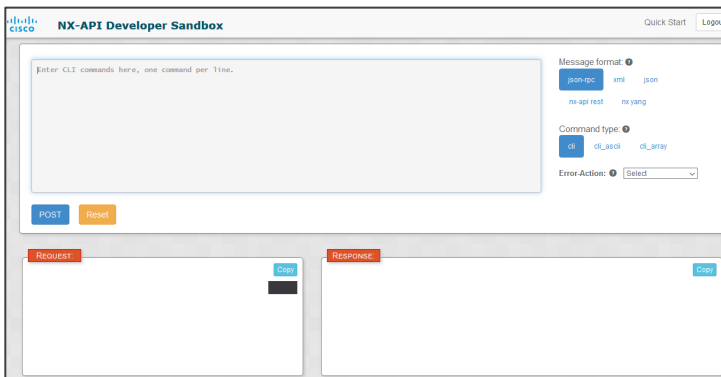
```
switch(config)# nxapi https port 8443
```

NX-API HTTP

HTTP port and HTTPS configuration



NX-API Sandbox



Web GUI for user to try out and get easy guidance on NX-API using http

- Commands are typed in to see format JSON-RPC, JSON, or XML
- User can send requests and receive responses
- Disabled by default



Demo



Configure APIs using Nexus Developer Sandbox



Examining XML and JSON for API



XML

Text-based format for describing data

The basic XML building block is an element that is defined by tag

An element consists of an opening tag, its attributes, any content, and a closing tag

Each element has a beginning and an ending tag



XML Request

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<ins_api>
```

```
  <version>0.1</version>
```

```
  <type>cli_show</type>
```

```
  <chunk>0</chunk>
```

```
  <sid>session1</sid>
```

```
  <input>show switchname</input>
```

```
  <output_format>xml</output_format>
```

```
</ins_api>
```

XML Response

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>Boston</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

JSON

Alternative to XML

It is a text-based format

It allows nested elements to provide hierarchy

The elements can be objects, arrays, or key/value pairs



XML Example

```
<switches>
```

```
  <switch name="BostonDC-switch01">
```

```
    <hostname>BostonDC-switch01</hostname>
```

```
    <mgmt-interface>mgmt0</mgmt-interface>
```

```
    <mgmt-ip>192.168.1.11</mgmt-ip>
```

```
    <mgmt-mask>255.255.255.0</mgmt-mask>
```

```
    <mgmt-gw>192.168.1.1</mgmt-gw>
```

```
  </switch>
```

```
</switches>
```

JSON Response

```
{
  "ins_api": {
    "type": "cli_show",
    "version": "0.1",
    "sid": "eoc",
    "outputs": {
      "output": {
        "body": {
          "hostname": "Boston"
        },
        "input": "show switchname",
        "msg": "Success",
        "code": "200"
      }
    }
  }
}
```

NX-API Elements to Specify a CLI Command

Request Element	Description
version	Specifies the NX-API version
type	Specifies the type of command to be executed. You can use cli_show , cli_show_ascii , cli_conf , or bash
chunk	Some show commands can return a large amount of output. For the NX-API client to start processing the output before the entire command is executed, NX-API supports output chunking for show commands
roll_back	Specifies the configuration roll-back options. You can use stop-on-error , continue-on-error , or rollback-on-error



NX-API Elements to Specify a CLI Command

Request Element	Description
validate	This element allows you to validate the commands before you apply them on the switch
sid	Specifies the session ID. This element is valid only when the response message is chunked
input	Input can be one command or multiple commands
lock	Allows you to specify an exclusive lock on the configuration whereby no other management or programming agent will be able to modify the configuration if this lock is held
output_format	The output format, which could be xml or json



XML or JSON Format Response Elements

Response Element	Description
version	Specifies the NX-API version
type	Specifies the type of command to be executed
sid	Session ID of the response. This element is valid only when the response message is chunked
outputs	Tag that encloses all command outputs
output	Tag that encloses the output of a single command output




XML or JSON Format Response Elements

Response Element	Description
input	Tag that encloses a single command that was specified in the request
body	Body of the command response
code	Error code returned from the command execution. NX-API uses standard HTTP error codes
msg	Error message associated with the returned error code



```
$ curl --user admin:cisco123 -X POST -d @api.xml -H "Content-Type: text/xml" http://192.168.1.2/ins
```

```
<ins_api>
  <type>cli_show</type>
  <version>1,2</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show hostname</input>
      <msg>success</msg>
    <code>200</code>
  </output>
</outputs>
</ins_api>
```



```
<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>show hostname</input>
  <output_format>xml</output_format>
</ins_api>
```

--user admin:cisco123: The basic HTTP authentication is used. The value for this argument is username:password

-X POST: The request uses the HTTP POST method

-d @api.xml: The data that was sent to the switch in the HTTP body, is specified in the api.xml file

-H "Content-Type: text/xml": The content type for this request is XML

Curl Arguments




```
$ curl --user admin:cisco123 -X POST -d @api.json -H "Content-Type: text/json" http://192.168.1.2/ins
```

```
{
  "ins_api": {
    "type": "cli_conf",
    "version": "1,2",
    "sid:: "eoc",
    "outputs": {
      "output": {
        "msg": "Success",
        "code": "200",
        "body": {
        }
      }
    }
  }
}
```



```
{
  "ins_api": {
    "version": "1.2",
    "type": "cli_conf",
    "chunk": "0",
    "sid": "1",
    "input": "hostname Boston_Switch",
    "output_format": "json"
  }
}
```

--user admin:cisco123: The basic HTTP authentication is used. The value for this argument is username:password

-X POST: The request uses the HTTP POST method

-d @api.xml: The data that was sent to the switch in the HTTP body, is specified in the api.xml file

-H "Content-Type: text/xml": The content type for this request is XML

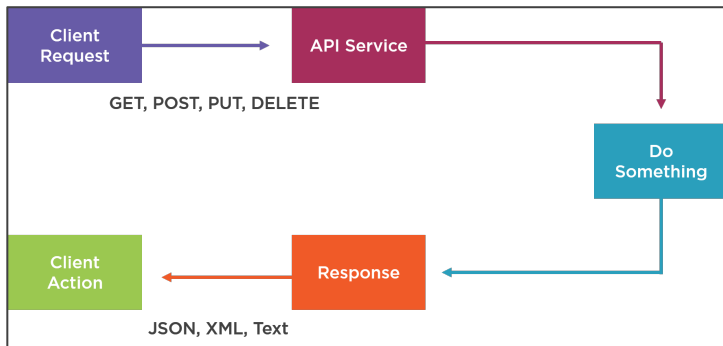
Curl Arguments



NX-API REST



NX-API REST



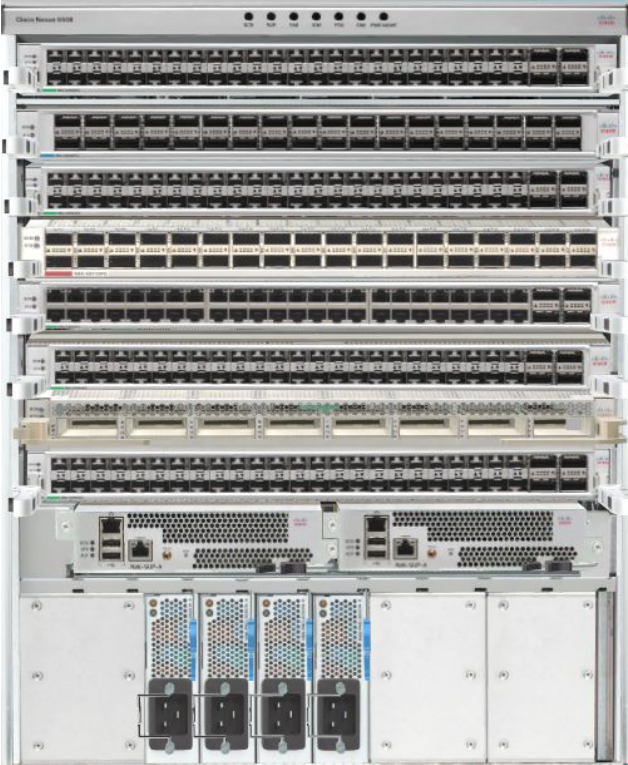
NX-API REST brings Model Driven Programmability (MDP) to standalone (non-APIC-based fabric) Nexus switches

Communicates over HTTP

Uses common HTTP verbs (GET, PUT, POST, DELETE) to extract data from a server

Returns data in different format per request (XML, JSON)

REST-API



Easy to use

Github code repository, code management, publicly available

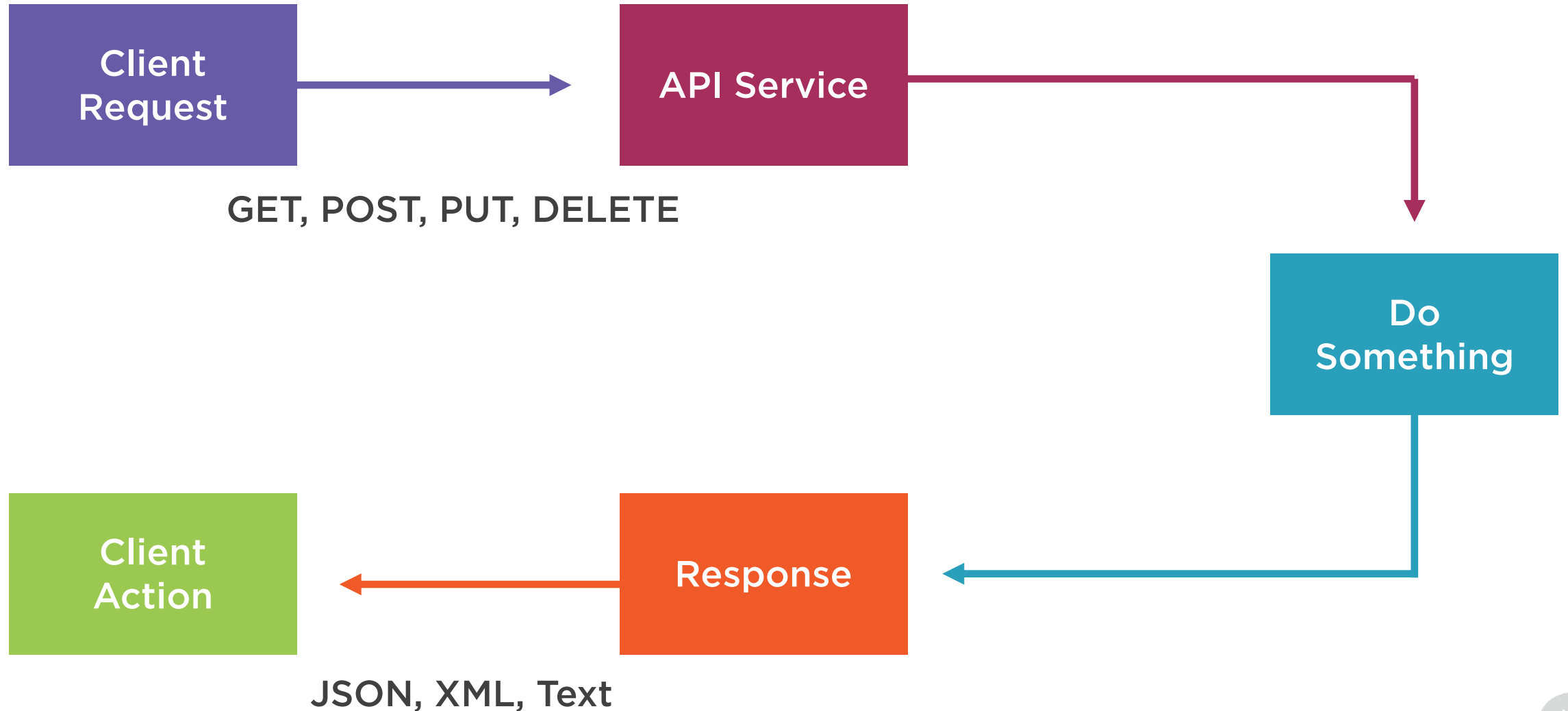
Lots of developers use

Many uses for REST-APIs

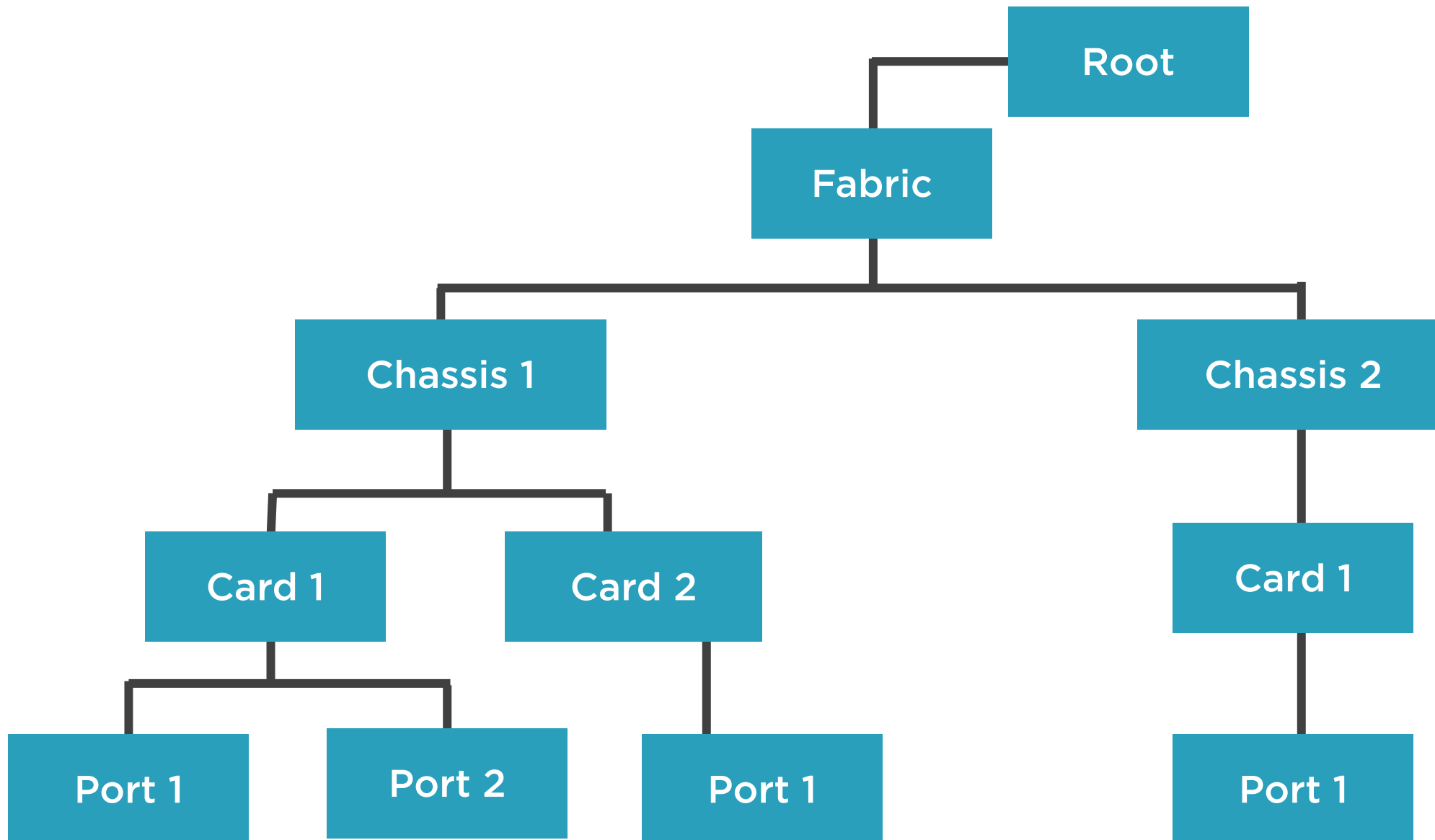
- Mobile apps
- Console apps
- Web apps



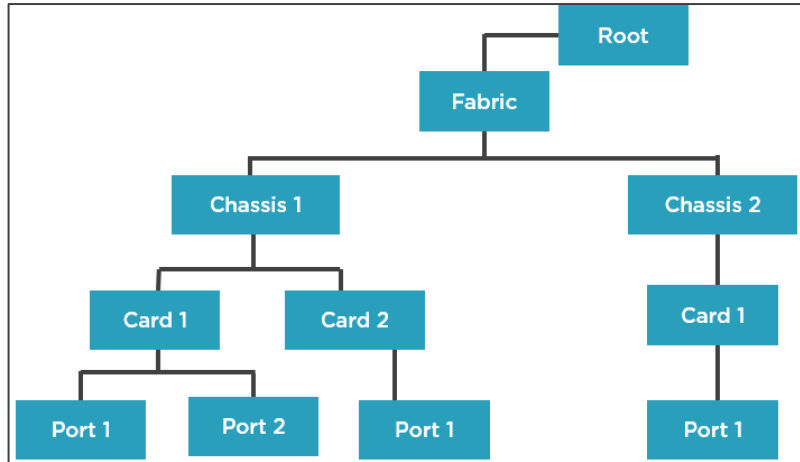
REST – Request and Response



NX-API-REST Managed Objects



REST-API



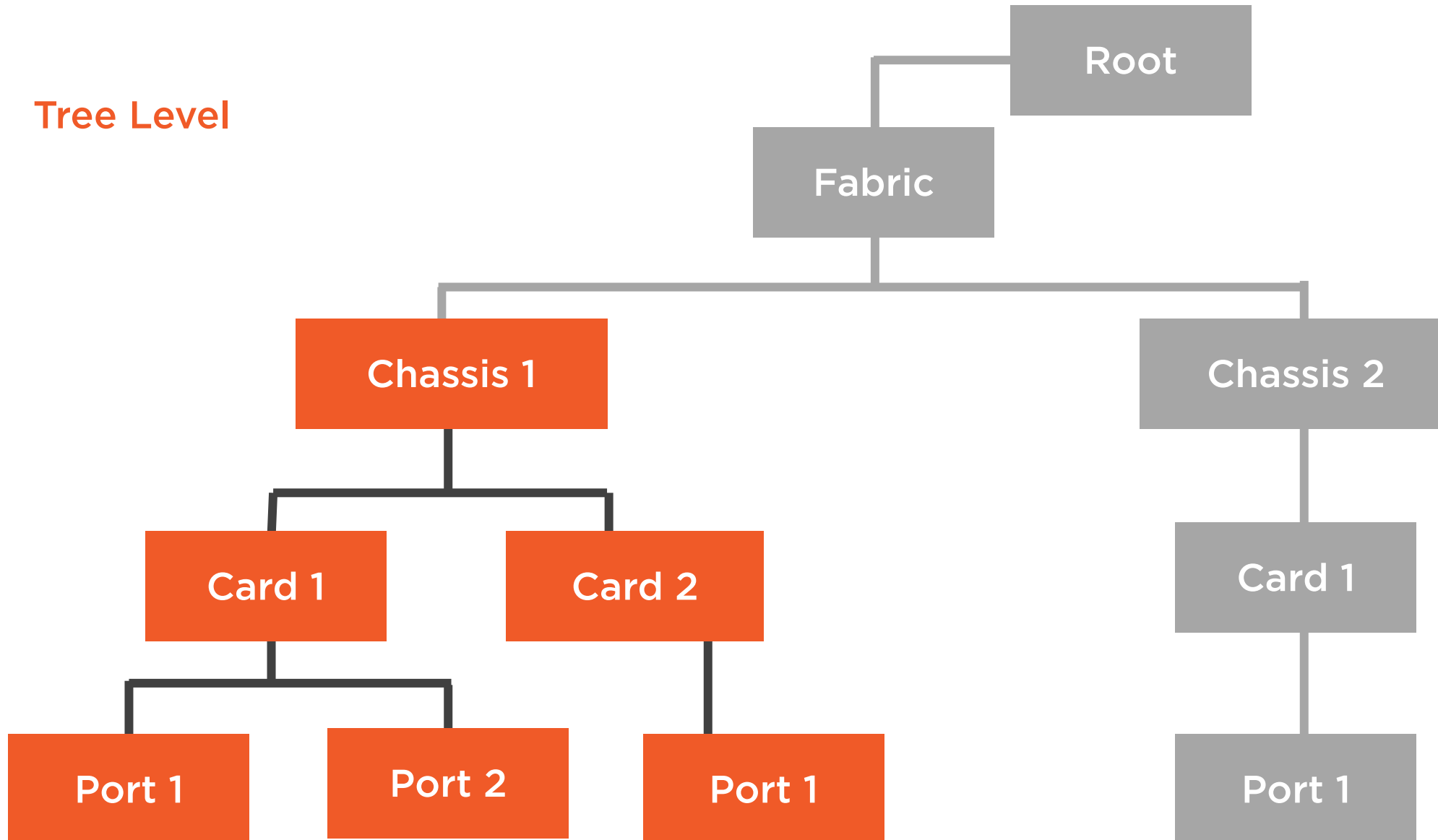
NX-API REST Queries

- Tree
- Class
- Object



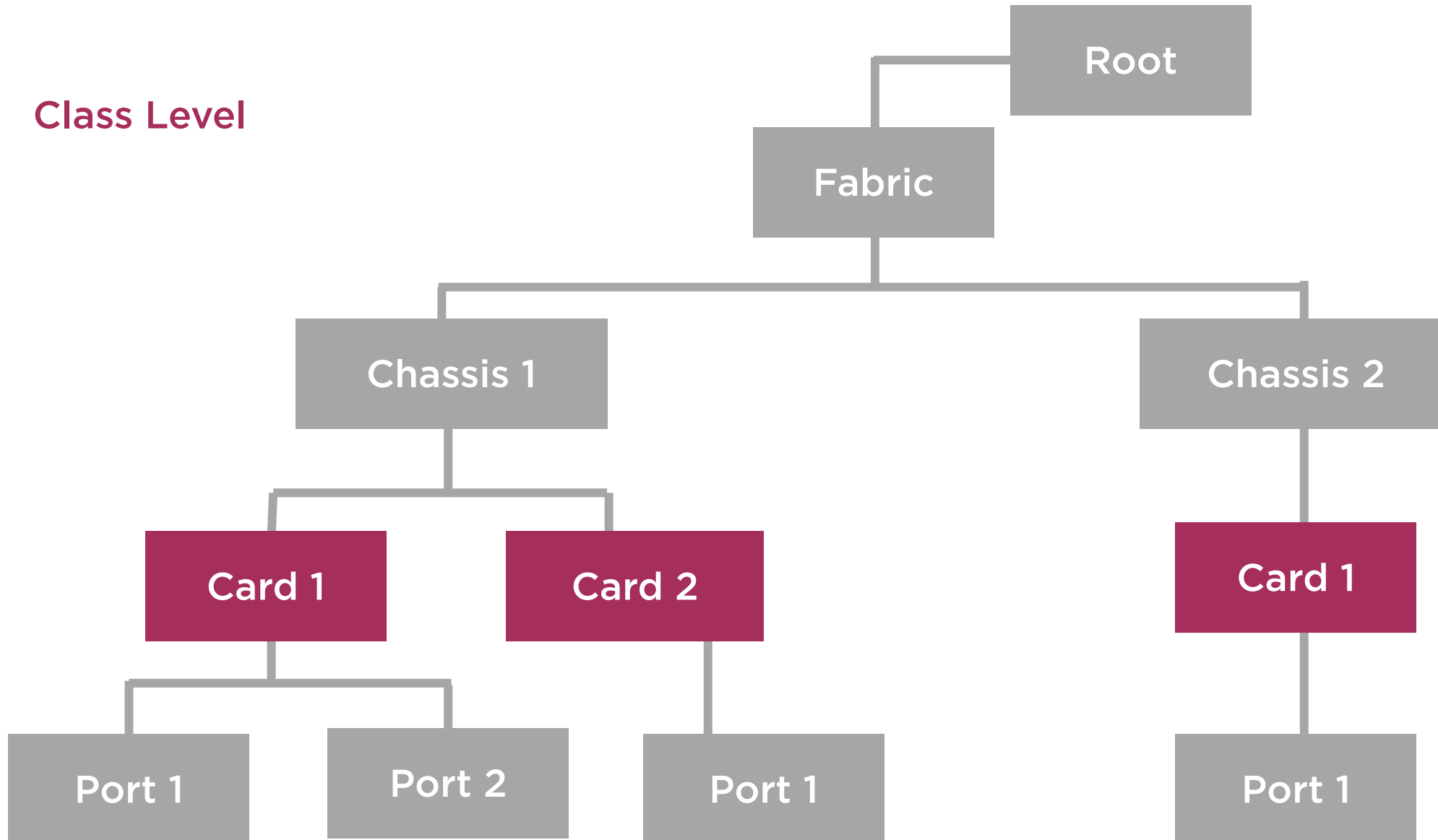
NX-API-REST Tree Level Query

Tree Level



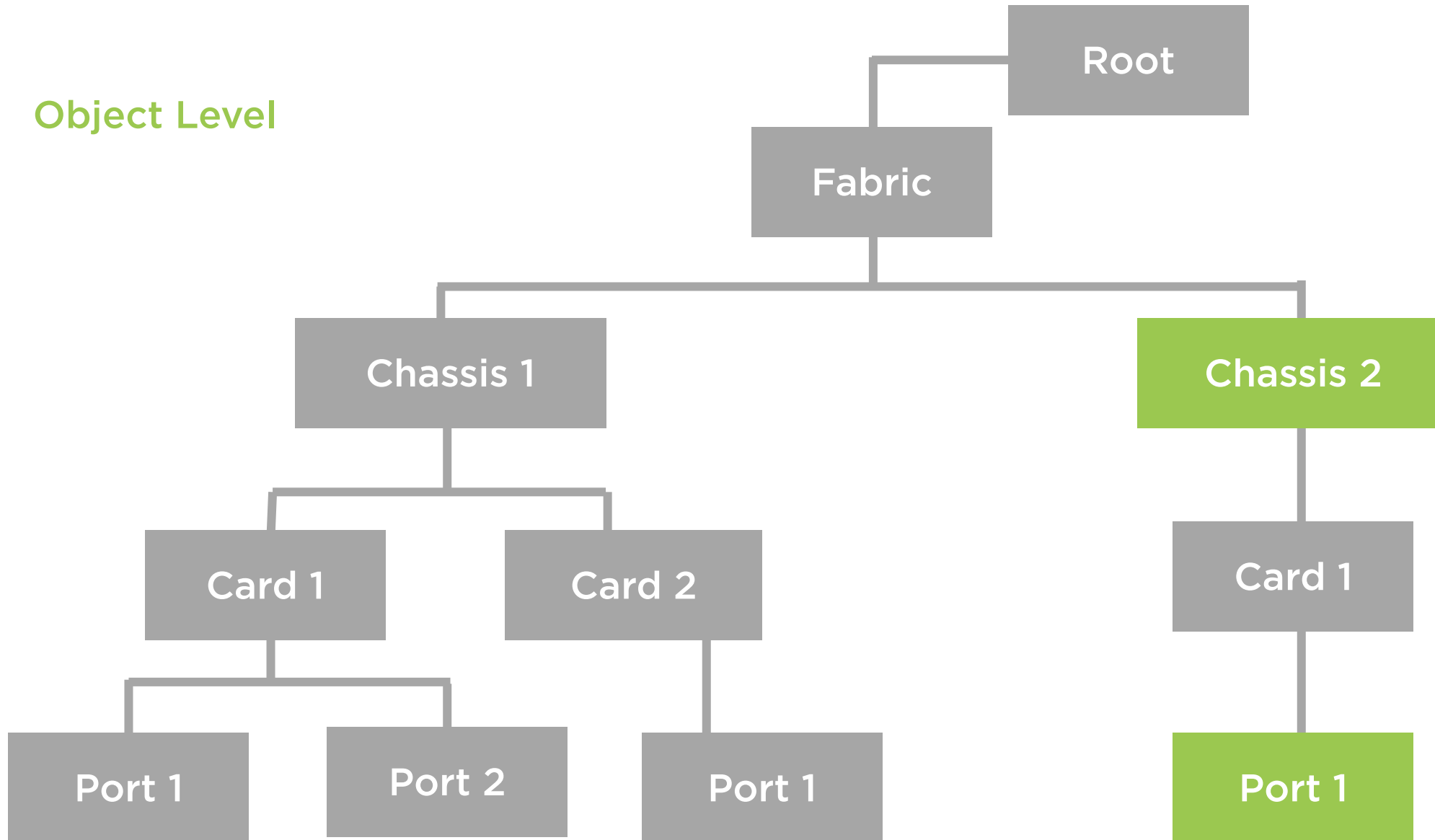
NX-API-REST Class Level Query

Class Level



NX-API-REST Object Level Query

Object Level



REST – Requests

Verb	CRUD	Description
POST	Create	Creates a new entity or service on the REST server
GET	Read	Used to retrieve information from the REST server
PUT	Update	Updates resources in the REST server API
DELETE	Delete	Deletes the specified resource that is invoked

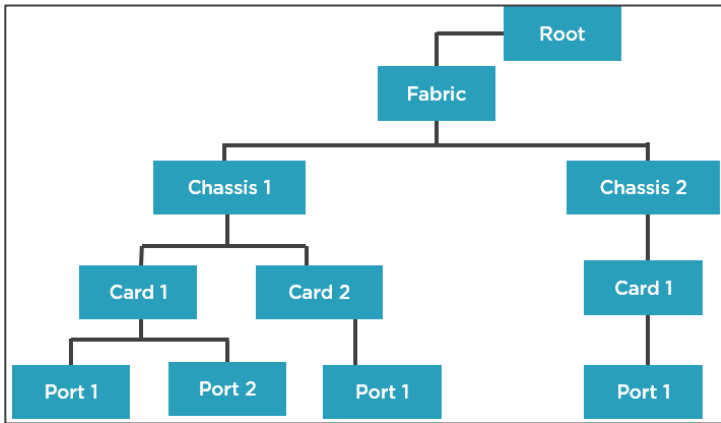


REST – Responses

Code	Message	Reason
200	OK	All looks good
201	Create	New resource created
400	Bad Request	Request was invalid
401	Unauthorized	Authentication missing or incorrect
403	Forbidden	Request was understood but not allowed
404	Not Found	Resource not found
500	Internal Server Error	Something wrong with the server
503	Service Unavailable	Server is unable to complete request



REST API-Automation



Client begins sending requests when the client is ready to make the transition to a new state

The URI for the web API, such as <http://example.com/resources/>

The type of the data that is supported by the web API. This type is often JSON, but can be any other Internet media type



Cisco NX-API-REST Message

`http(s)://[switch:port]/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

System: System identifier; an IP address or DNS-resolvable host name



Cisco NX-API-REST Message

`http(s)://switch:port/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

Specifies that the message is directed to the API



Cisco NX-API-REST Message

`http(s)://switch:port/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

mo|class: Specifies whether the target of the operation is an MO or an object class



Cisco NX-API-REST Message

`http(s)://switch:port/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

dn—Specifies the distinguished name (DN) of the targeted MO

className—Specifies the name of the targeted class



Cisco NX-API-REST Message

`http(s)://switch:port/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

method: Optional method being invoked on the object; applies only to POST requests



Cisco NX-API-REST Message

`http(s)://switch:port/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

json|xml: the encoding format (JSON or XML)



Cisco NX-API-REST Message

`http(s)://switch:port/api/[mo|class]/[dn|class][:method].[xml|json]?{options}`

?options—(Optional) Specifies one or more filters, selectors, or modifiers to a query



Demo



Configuring Cisco NX-API REST URIs
using Visore and Postman



Ansible and Puppet Integration



Ansible Overview

```
- name: feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
# Ensure ospf is enabled
  - nxos_feature: feature=ospf
    state=enabled host={{ inventory_hostname}}
```

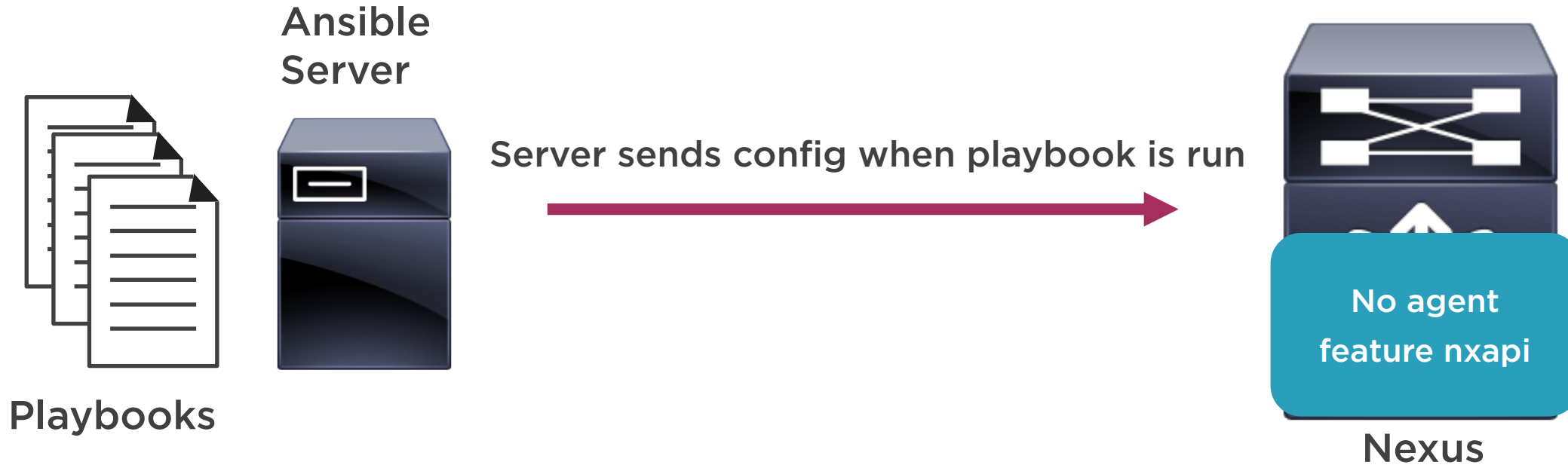
Ansible is an open-source software platform used for configuring and managing compute and switching infrastructure

Uses playbooks, which are Ansible's configuration, deployment, and orchestration language

Used to automate the configuration of compute and switching resources



Ansible



Ansible uses an agentless push model
Configuration files (playbooks) use YAML
Can configure using CLI (SSH) or NX-API



Ansible Overview

```
- name: feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
# Ensure ospf is enabled
  - nxos_feature: feature=ospf
    state=enabled host={{ inventory_hostname}}
```

Uses push-based model

Scripts run on the management server, connect to the managed device, and execute tasks

Makes extensive use of Python

Requires SSH and Python support on the target node, but you can also easily extend Ansible to use any API



Using Ansible with NX-OS

Ansible is a very straightforward and powerful tool for intent-based network automation

All that you need to get started are playbooks, a server configuration file, and an inventory file

Does not require you to install a software agent on the target node (server or switch) to automate the device.



Key Ansible Concepts

```
- name: Feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    # Ensure ospf is enabled
    - name: Feature: Feature-ospf
      state-enabled host={{ inventory_hostname }}
```

Playbooks: A list of tasks that run in sequence across one or more hosts. Each task can also run multiple times with different variables

```
- name: Feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    # Ensure ospf is enabled
    - name: Feature: Feature-ospf
      state-enabled host={{ inventory_hostname }}
```

Inventory: Information about hosts. Describes to which groups a host belongs, and the properties those groups and hosts have

```
- name: Feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    # Ensure ospf is enabled
    - name: Feature: Feature-ospf
      state-enabled host={{ inventory_hostname }}
```

Templates: Generate configuration files from values that are set in various inventory properties

```
- name: Feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    # Ensure ospf is enabled
    - name: Feature: Feature-ospf
      state-enabled host={{ inventory_hostname }}
```

Roles: Provide a way to encapsulate common tasks and properties for reuse



Ansible Playbook YAML

```
- name: feature testing
```

```
  hosts: all
```

```
  connection: local
```

```
  gather_facts: no
```

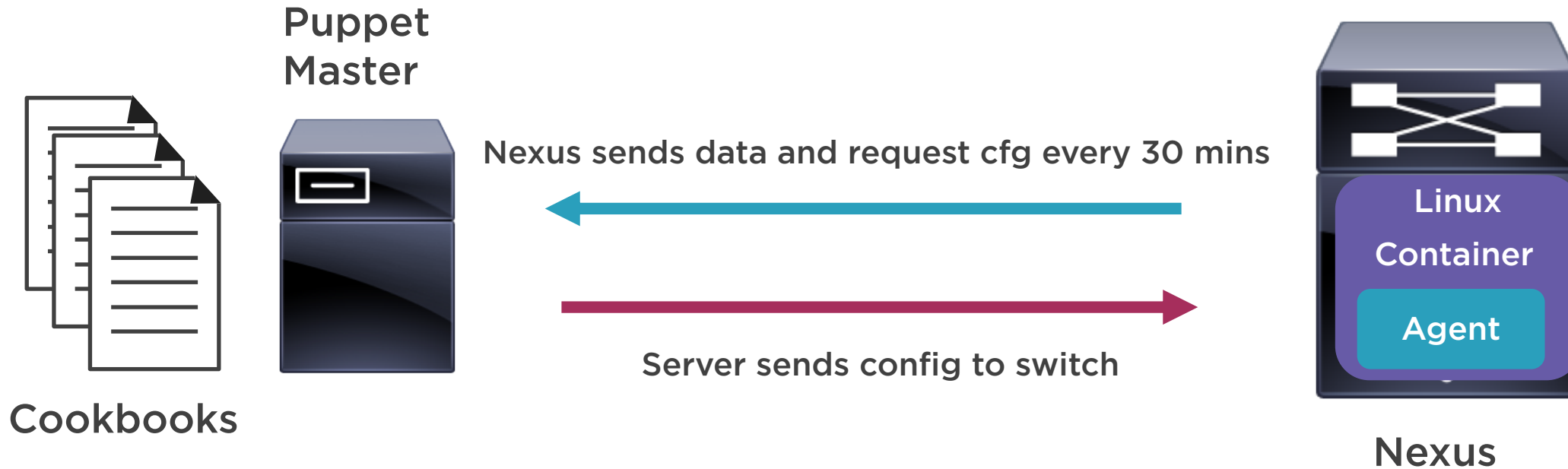
```
  tasks:
```

```
# Ensure ospf is enabled
```

```
  - nxos_feature: feature=ospf
```

```
    state=enabled host={{ inventory_hostname }}
```

Puppet Integration Overview



Puppet use a pull model (agent/client pulls from server)

Agent/client lives in Linux container (directly in Bash on 3K & 9K)



Puppet System States



Puppet models the desired system states, enforces those states, and reports any variances so you can track what Puppet is doing



Puppet uses a declarative resource-based language, which means that a user describes the desired final state



If we have a pre-defined configuration that every new switch should receive, the intent-based automation can automate them



Open-source Puppet can be appropriate if you are managing several devices; large environments should use Puppet Enterprise



Phases of Puppet Integration



Define: With Puppet declarative language we can design a graph of relationships between resources within reusable modules



Simulate: With this resource graph, Puppet is unique in its ability to simulate deployments, enabling testing changes without disruption

```
- name: feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    # Ensure app is enabled
    - name: feature: feature-on-off
      state-enabled host={{ inventory_hostname }}
```

Enforce: Puppet compares our system to the desired state as we define it, and automatically enforces it to the desired state

```
- name: feature testing
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    # Ensure app is enabled
    - name: feature: feature-on-off
      state-enabled host={{ inventory_hostname }}
```

Report: Puppet Dashboard reports track relationships between components and all changes



Puppet Agents



The Puppet program is called a manifest

Deploy the manifest on a Nexus 9000, it translates into network configuration settings and commands for collecting data

Puppet agent

- Runs on the managed device (node)

Puppet master

- Runs on a separate dedicated server

Using Puppet with Cisco NX-OS Devices



Bash shell: The native Linux environment underlying NX-OS. Disabled by default. It can be used on Nexus 3000 and 9000 switches for Puppet agent installation



Guest shell: The secure Linux container environment. Enabled by default in most platforms that support it. It can be used on Nexus 3000 and 9000 switches for Puppet agent installation



Open Agent Container (OAC): A 32-bit CentOS-based container that is running Puppet agent software on Nexus 5000, Nexus 6000, and Nexus 7000 switches



Puppet and UCS

```
ucsm_macpool{'macpoolBoston':  
  policy_name => "macpool_Bstnsrv",  
  descr => "servers",  
  to => "00:05:C1:00:44:10",  
  r_from => "00:05:C1:00:44:01",  
  ip => "192.168.1.14",  
  username => "admin",  
  password => "password",  
  state => "present",  
}
```

Summary



Cisco NX-OS supports intent-based automation through various tools

- NX-API
- NX-API REST
- Ansible and Puppet

